



# THÈSE

En vue de l'obtention du

**DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE**

**Délivré par** l'Université Toulouse III - Paul Sabatier  
**Discipline ou spécialité:** Informatique

---

**Présentée et soutenue par** Fabrizio BORGIA  
**Le 30 Mars 2015**

**Titre:** Informatisation d'une forme graphique des Langues des Signes:  
application au système d'écriture SignWriting

---

## JURY

Philippe JOLY - Président du jury  
Annelies BRAFFORT - Rapporteur de thèse  
Alessandro MEI - Membre du jury  
Guido PROIETTI - Membre du jury

---

**Ecole doctorale:** Mathématiques, Informatique, Télécommunications  
**Unité de recherche:** Institut de Recherche en Informatique de Toulouse  
**Directeurs de Thèse:** Michel DAYDE, Maria DE MARSICO  
**Rapporteurs:** Alexey KARPOV, Annelies BRAFFORT

*“And they shall know no fear.”*  
The Emperor of Mankind

# Acknowledgments

The road leading to the achievement of a Ph.D. can be quite harsh. The guidance of my supervisors, the help of colleagues and friends, and the support of my family and my girlfriend made the present work possible, keeping me on my road. Each of them has helped to shape it, someone in a small way, someone in a decisive way. Therefore I would like to thank:

My supervisor, Prof. Patrice Dalle (1948-2014), for welcoming me in Toulouse, for inspiring me and for his generous guidance during these past four years. Thank you, Patrice, for your invaluable teachings, and for a few words that I am never going to forget.

My supervisor for the final part of my Ph.D., Prof. Michel Dayde, for his presence and for his precious help.

My co-supervisor, Prof. Maria De Marsico, for everything. Thank you for many insightful discussions and suggestions, and for being my primary resource for getting my questions answered, whenever even Google cannot. A much bigger thank you for believing me, for your patience, for your humor and for being there, always.

The reviewers of the present work: Prof. Annelies Braffort, Prof. Alexey Karpov and Prof. Kent L. Norman for their insightful comments.

The directors of the Ph.D. schools in Rome and in Toulouse: Prof. Alessandro Mei (“Sapienza” University of Rome) and Prof. Jean-Michel Roquejoffre (Université Toulouse III - Paul Sabatier).

Elena Antinoro Pizzuto (1952-2011), for her contagious enthusiasm, for her activism towards the cause of the acknowledgement of Sign Language and for showing me the deep meaning of working with passion. Thank you for being the first person to believe in the possibility for me to become a Ph.D. and for actively working for it.

Claudia S. Bianchini, my sister-in-arms, for the fun we had working together and traveling around for our conferences, for guiding me and for helping me

with any linguistic matter, and with a big lot of other things. Thank you for your friendship, for cheering me up whenever things got crazy, and above all for stopping that ferry in Istanbul!

Valerie Sutton, the inventor of SignWriting, for “being the giant on whose shoulders I climbed”, for her unwavering enthusiasm and for her limitless energy. It was my honor to meet you, Valerie.

The research team of the ISTC-CNR at the State Institute for Deaf of Rome, for helping me to approach the Deaf Culture and for being my very first teachers in Sign Language. Thanks for all the insightful meetings and for your remarkable help during my studies and during the design and testing of most digital artifacts presented in this work.

Arianna Testa, because, through her excellent job as a Sign Language interpreter, she helped in producing a usability test for deaf users.

All the graduating students who worked at the digital artifacts presented in this work: Lorenzo Trovarelli, Nigel Pillaha, Marco Rotiroti, Luca Franciosi, Davide Della Valle, Luca Fochetti and Riccardo Coccia. Thank you for our discussions, for sharing your thoughts and for your precious help.

Martine Labruyère and Alessandra Sabatini, for their invaluable help against countless deadly administrative issues both in France and in Italy.

The Bianchini-Marbot family: Marco L. Bianchini for the accurate reading of most of my papers, and for his precious advice on the academic world, and René Marbot for the hospitality and the company in the city of Paris.

The Serco group, in particular Francesco Ferrante and Fabrizio Giordano for their trust, their advice and for putting up with the frequent schedule changes that may occur in the life of a worker which is finishing his Ph.D.

---

My mother and my father, for my life as it is. Because I have so many things to be grateful for, that I have been looking at this sheet for two hours, without knowing where to start. I know they have been doing their best for me, every day, since that 25 June 1985, and despite they do not share all of my choices, they support me and they put up with no matter what.

My sister Barbara, for the huge imaginary of stories, names and characters that we share. Thank you for the happiness you bring, and the silence that you take away.

My grandparents: Maria, Ferdinando, Lucia and Nello, for the time we spent together, for the pride that I always read in their eyes, and because the thought of them is one of the things that drives me to never give up.

Chiara, for being my North. For backing me up with love and empathy through all these years. For the adventures we share, for the sharpness of her thinking, and the kindness of her words. Because in my darkest moments she has never “winced nor cried aloud”.

My friends Jacopo and Marco, for being there, no matter what, and for the countless hours of gaming we have been sharing together since our early years.

My friend Silvia, for making me laugh, and for being able to get me back with my feet on the ground when I’m wandering too far.



# Abstract

## Abstract en Français

Les recherches et les logiciels présentés dans cette étude s'adressent à une importante minorité au sein de notre société, à savoir la communauté des sourds. Selon l'Organisation Mondiale de la Santé, cette minorité compte plus de 278 millions de personnes dans le monde. De nombreuses recherches démontrent que les sourds ont de grosses difficultés avec la langue vocale (LV – Anglais, Chinois, etc.) ce qui explique que la plupart d'entre eux préfèrent communiquer en Langue des Signes (LS). Du point de vue des sciences de l'information, les LS constituent un groupe de minorités linguistiques peu représentées dans l'univers du numérique. Et, de fait, les sourds sont les sujets les plus touchés par la fracture numérique.

Cette étude veut donc être une contribution pour tenter de combler ce fracture numérique qui pénalise les sourds. Pour ce faire, nous nous sommes principalement concentrés sur l'informatisation de SignWriting, qui constitue l'un des systèmes les plus prometteurs pour écrire la LS. Concrètement, SignWriting est un système d'écriture qui utilise des symboles pour représenter les configurations des mains, les mouvements et les mimiques du visage de la LS.

Nos travaux visent donc à projeter et élaborer un système pour développer la production et l'utilisation des ressources en LS écrites avec SignWriting. Ce système a été baptisé: SignWriting-oriented resources for the deaf (SWord). Le but final de SWord est de rendre SignWriting effectivement exploitable par les sourds aussi bien en tant que moyen de communication qu'en tant que support d'apprentissage notamment dans le domaine du numérique. SWift, un éditeur numérique permettant la création de ressources numériques en LS écrites avec SignWriting, a été le premier logiciel à être inclus dans SWord. Dans la présente étude, nous souhaitons illustrer une série de fonctions mises à jour de SWift, comme par exemple la possibilité de composer des histoires entières en LS. En outre, pour évaluer la fiabilité et la facilité d'utilisation de l'application, nous avons organisé une session de tests sur un échantillon de ses principaux usagers, à savoir des personnes qui connaissent et utilisent SignWriting. Etant donné que cet échantillon est composé en majeure partie de sourds, nous avons adapté l'approche des tests de façon à ce qu'elle soit valable aussi bien pour les utilisateurs de la LS que ceux de la LV. Pour la réalisation du test, nous avons adapté une méthode classique d'évaluation de

la facilité d'utilisation, à savoir le Think-Aloud Protocol (TAP) ainsi qu'un questionnaire de satisfaction très répandu, à savoir le Questionnaire for User Interaction Satisfaction (QUIS).

En dépit des efforts accomplis, nous avons constaté que les éditeurs numériques pour SignWriting, comme par exemple SWift (et de nombreux autres) sont encore bien loin d'offrir à l'utilisateur une interface en mesure d'imiter la simplicité de l'écriture manuscrite. C'est la raison pour laquelle nous avons approfondi nos recherches afin de projeter une nouvelle génération d'éditeur pour SignWriting, en mesure d'épargner à l'utilisateur d'éventuelles obligations liées au click, au glissé-déposé, à la recherche et à la navigation sur l'Interface Utilisateur (IU) pendant le processus de composition d'un signe. Notre objectif est de réaliser un mode d'interaction se rapprochant le plus possible de la méthode papier-stylo dont se servent habituellement les êtres humains pour écrire ou dessiner.

Cet objectif représente le noyau de notre étude. Pour pouvoir réaliser cette nouvelle génération d'éditeur, nous avons conçu et élaboré un logiciel préposé à la conversion électronique d'images scannées contenant des symboles manuscrits ou imprimés en textes numériques dans SignWriting. Cette technique est dénommée SignWriting Optical Glyph Recognition (SW-OGR) dans la présente étude. Apparemment SW-OGR a beaucoup de points en commun avec les techniques de Optical Character Recognition (OCR) pour LV, qui sont largement entérinées par la littérature sur la vision par ordinateur. Par conséquent, après avoir recueilli un bon nombre de textes manuscrits dans SignWriting, en fonction de leur caractéristique picturale, nous avons étudié les similitudes possibles entre le SW-OGR et les techniques à la pointe de l'OCR pour les caractères manuscrits de la langue arabe, indienne et chinoise. Même si ces techniques adoptent des approches similaires, fondées sur des techniques de reconnaissance des formes et de l'Apprentissage Automatique, leur application dépend souvent de la notation spécifique qui doit être reconnue.

Dans le cas présent, nous n'avons pas pu suivre le même parcours pour la conception du moteur SW-OGR et nous avons dû élaborer nos propres procédures de reconnaissance. Un choix qui a été dicté par les caractéristiques de SignWriting. Primo parce que, contrairement à de nombreux alphabets, SignWriting offre une série de symboles de l'ordre de dizaines de milliers. Secundo parce que, habituellement, l'écriture à la main étant relativement imprécise, chaque symbole peut être dessiné de plusieurs façons différentes (généralement de l'ordre de dizaines de façons différentes), comme n'importe qu'elle notation manuscrite.

Par conséquent, il est évident que n'importe quelle formation pour une approche basée sur l'apprentissage automatique prendrait un temps exagéré et, surtout, demanderait un nombre considérable de modèles. Il y a également une autre raison qui complique l'application des techniques de reconnaissance des formes: l'absence de règles fixant le nombre, le type et la position des symboles dans un signe écrit avec SignWriting. En effet, la liberté de composition, qui est l'une des caractéristiques faisant de SignWriting un in-



---

strument très utile pour l'écriture au quotidien en LS, complique beaucoup la modélisation de la notation.

Pour toutes les raisons mentionnées ci-dessus, nous avons donc projeté et mis en place SW-OGR de façon à effectuer la reconnaissance des textes dans SignWriting en travaillant exclusivement sur les caractéristiques géométriques et topologiques des symboles et leurs relations topologiques. Nous nous sommes également appuyés sur des informations contextuelles, comme par exemple la connaissance de l'organisation de l'alphabet de SignWriting.

Dans la présente étude nous présentons également nos recherches sur l'alphabet de SignWriting visant à d'identifier les critères géométriques permettant de classer ses symboles, la conception de la procédure de reconnaissance, l'élaboration et la mise à l'essai du moteur SW-OGR. La première partie de la procédure vise à repérer les formes bases au sein des fragments présents dans le texte écrit (cercles, polygones, etc.). C'est la seule étape au cours de laquelle nous pouvons exploiter les procédures d'apprentissage automatique existantes. Les formes détectées et leurs caractéristiques sont utilisées pour inférer la région du corps et le type d'élément non-anatomique (mouvement, contact, etc.) que le fragment peut décrire.

A partir de là, les critères identifiés correspondent à une série de vérifications effectuées pour reconnaître les caractéristiques du symbole. Les résultats de ces vérifications sont ensuite utilisés pour construire un code, qui est lui-même utilisé pour identifier le symbole.

Le moteur SW-OGR a vocation à accomplir une double fonction: en premier lieu, il peut être intégré dans un éditeur de SignWriting existant, tel que SWift, afin de fournir un support immédiat à l'écriture manuscrite et de rendre le processus de composition beaucoup plus rapide et pratique pour un usage quotidien. A signaler que depuis le début de nos travaux, nous étions conscients de la présence (et des dimensions importantes) d'un certain nombre de corpus manuscrits en SignWriting recueillis par diverses communautés dans le monde. Cette documentation est un bien précieux qui pourrait être beaucoup plus utile encore si elle était numérisée. En second lieu, SW-OGR est donc en mesure de numériser ces corpus de façon « intelligente », à savoir qu'il ne se limite pas à effectuer un simple « balayage » des documents mais, à travers la reconnaissance, il est en mesure de recueillir toutes les informations sur le rapport entre les signes et les symboles qui les composent, et permet ainsi à la communauté scientifique de réaliser tout type d'analyse linguistique sur des séries de données complètes en SignWriting.

## Abstract in English

The studies and the software presented in this work are addressed to a relevant minority of our society, namely deaf people. According to the World Health Organization, such “minority” currently counts more than 278 million people worldwide. Many studies demonstrate that, for several reasons, deaf people experience significant difficulties in exploiting a Vocal Language (VL - English, Chinese, etc.). In fact, many of them prefer to communicate using Sign Language (SL). As computer scientists, we observed that SLs are currently a set of underrepresented linguistic minorities in the digital world. As a matter of fact, deaf people are among those individuals which are mostly affected by the digital divide.

This work is our contribution towards leveling the digital divide affecting deaf people. In particular, we focused on the computer handling of SignWriting, which is one of the most promising systems devised to write SLs. More specifically, SignWriting is a writing system which uses visual symbols to represent the handshapes, movements, and facial expressions of SLs.

Our efforts aim at designing and implementing a framework to support the production and the use of SignWriting-oriented resources for the deaf (SWord). The ultimate purpose of SWord is to make SignWriting effectively exploitable as a communication mean and a learning support for deaf people, especially in the digital world. The first software to be included into the SWord framework was SWift, a digital editor which allows the creation of digital SL resources written in SignWriting. In this work, we present a number of upgraded features of SWift, such as the possibility to write signed stories. Moreover, to assess the reliability and the usability of such application, we conducted a test session with its main target users, i.e. people which are proficient in SignWriting. Since our sample was mainly composed by deaf people, we adapted our testing approach to be viable for both SL and VL users. In order to perform the assessment, we adapted a popular usability testing methodology, namely the Think-Aloud Protocol (TAP), and a widespread customizable questionnaire, namely the Questionnaire for User Interaction Satisfaction (QUIS).

Despite our efforts, we observed that SignWriting digital editors, such as SWift (and many others), are still far from granting the user an interface which is able to emulate the simplicity of handwriting. For this reason, we continued working towards the possibility to design a new generation of SignWriting editors, able to lift the user of any burden related to clicking, dragging, searching, browsing on the User Interface (UI) during the composition process of a sign. Our aim is to implement an interaction style which is as similar as possible to the paper-pencil approach that humans normally use when writing or drawing.

The main part of the present work deals with this latter goal. To make such new generation of editors feasible, we designed and implemented a software application whose purpose is to operate the electronic conversion of scanned images containing handwritten or printed SignWriting symbols

---

into machine-encoded SignWriting texts. Such technique is referred to as the SignWriting Optical Glyph Recognition (SW-OGR) in the present work. Apparently, the SW-OGR topic shares much in common with the Optical Character Recognition (OCR) techniques for VLS, which are well consolidated within the computer vision literature. More specifically, after gathering a fair number of handwritten SignWriting texts, and given their pictorial nature, we investigated the possible similarities between the SW-OGR and the modern OCR techniques employed to perform the recognition of Arabic, Indian and Chinese handwritten characters. Even if such techniques adopt similar approaches, based on pattern recognition and machine learning, their implementation may often depend on the specific notation to be recognized. In our case, we could not follow a similar line for the design of the SW-OGR Engine, and we were forced to devise our own recognition procedures. The reason for this choice is the very particular nature of SignWriting. First of all, unlike most alphabets, SignWriting features a number of symbols which is in the order of the tens of thousands. Moreover, since handwriting is usually quite inaccurate, each symbol can be drawn in a number (usually in the order of tens) of different ways, as in any handwritten notation. As a consequence, it is evident that any training to enable a machine learning approach would require an unreasonable amount of time and, most of all, of training templates. A further reason makes the application of pattern recognition techniques very difficult, i.e. the total lack of rules regulating the number, type and position of the symbols within a SignWriting sign. In fact, the composition freedom, which is one of the features which make SignWriting a very handy tool for everyday SL writing, also makes the modeling of the notation very difficult.

For the above reasons, we designed and implemented SW-OGR to perform the recognition of SignWriting texts by only working with the geometric and topological features of the symbols, and with their topological relationships. We also relied on context-dependent information, such as the knowledge of the organization of the SignWriting alphabet.

In this work we present the studies performed on the SignWriting alphabet in order to identify geometric criteria to classify its symbols, the design of the recognition procedure, and the development and testing of the SW-OGR engine. The starting points of the procedure aims at identifying base shapes within the fragments of the written text (circles, polygons, etc.). This is the only step where we can exploit existing machine learning procedures. Detected shapes and their features are used to infer the body area or kind of non-anatomical element (movement, contact, etc.) that the fragment may depict. From this point on, the identified criteria correspond to a series of checks performed to recognize the features of the symbol. The result of the checks are used to build a code, which is finally used to identify the symbol. The engine is intended to serve a twofold purpose: first of all, it can be embedded within existing SignWriting editors, such as SWift, in order to provide a prompt support for handwriting, and make the composition process much faster and comfortable for everyday use. In addition, since the

beginning of our work, we were aware of the presence (and of the considerable size) of a number of handwritten SignWriting corpora gathered from different communities around the world. Those corpora are an invaluable asset, and they could become even more useful if digitalized. SW-OGR is able to digitalize such corpora in a smart way, since it does not simply perform a “scan” of the documents, but, through the recognition, it is able to gather any information about the relationship between a sign and the symbols that compose it, thus allowing the research community to perform any kind of linguistic analysis on whole SignWriting datasets.

---

## Abstract in Italiano

Gli studi ed i software presentati in questo lavoro sono indirizzati ad una importante minoranza nella nostra società, ossia quella costituita dalle persone sorde. Attualmente, secondo la World Health Organization, tale “minoranza” è costituita da più di 278 milioni di persone in tutto il mondo. Molti studi dimostrano che, per diverse ragioni, le persone sorde incontrano serie difficoltà nell'utilizzo di una Lingua Vocale (LV - Inglese, Cinese, ecc.). Infatti, molti di loro preferiscono comunicare usando la Lingua dei Segni (SL). Da un punto di vista delle scienze dell'informazione, si può osservare che le LS costituiscono attualmente un gruppo di minoranze linguistiche poco rappresentate nel mondo digitale. Di fatto, le persone sorde sono tra gli individui maggiormente affetti dalla digital divide.

Il presente lavoro rappresenta il nostro contributo per ridurre la digital divide che colpisce le persone sorde. In particolare, ci siamo concentrati sull'informatizzazione di SignWriting, che è uno dei sistemi più promettenti ideati per scrivere la LS. Più specificamente, SignWriting è un sistema di scrittura che utilizza simboli per rappresentare le configurazioni delle mani, i movimenti e le espressioni facciali della LS.

I nostri sforzi mirano a progettare e implementare un framework per favorire la produzione e l'utilizzo di risorse in LS scritte con Signwriting, il nome del framework è SignWriting-oriented resources for the deaf (SWord). Il fine ultimo di SWord è di rendere SignWriting effettivamente sfruttabile come mezzo di comunicazione e come supporto per l'apprendimento per le persone sorde, soprattutto nel mondo digitale. Il primo software ad essere incluso in SWord è stato SWift, un editor digitale che permette la creazione di risorse digitali in LS scritte con SignWriting. In questo lavoro, presentiamo una serie di features aggiornate di SWift, come ad esempio la possibilità di comporre intere storie in LS. Inoltre, per valutare l'affidabilità e l'usabilità di tale applicazione, abbiamo condotto una sessione di test con un campione dei suoi principali utenti finali, ossia persone che conoscono ed usano SignWriting. Dato che il nostro campione è composto principalmente da persone sorde, abbiamo adattato il nostro approccio di test in modo da essere valido sia per gli utenti di LS che di LV. Al fine di eseguire il test, abbiamo adattato un popolare metodo per la valutazione dell'usabilità, ossia il Think Aloud Protocol (TAP), e un questionario personalizzabile molto diffuso, vale a dire il Questionario per la User Interaction Satisfaction (QUIS).

Nonostante i nostri sforzi, abbiamo osservato che gli editor digitali per Signwriting, come ad esempio SWift (e molti altri), sono ancora molto lontani dal fornire all'utente un'interfaccia che sia in grado di emulare la semplicità della scrittura a mano. Per questo motivo, abbiamo continuato a lavorare verso la possibilità di progettare una nuova generazione di editor per SignWriting, in grado di sollevare l'utente da eventuali oneri legati a click, drag-and-drop, ricerca e navigazione sull'Interfaccia Utente (UI) durante il processo di composizione di un segno. Il nostro obiettivo è realizzare uno stile di interazione che sia il più simile possibile al metodo carta-e-penna che gli esseri umani

normalmente utilizzano durante la scrittura o il disegno.

La parte principale del presente lavoro affronta quest'ultimo obiettivo. Per rendere realizzabile questa nuova generazione di editor, abbiamo progettato e implementato un software il cui scopo è quello di operare la conversione elettronica di immagini scansionate contenenti simboli scritti a mano o stampati in testi digitali in Signwriting. Nel presente lavoro, tale tecnica viene definita come SignWriting Optical Glyph Recognition (SW-OGR).

Apparentemente, SW-OGR ha molto in comune con le tecniche di riconoscimento ottico dei caratteri (OCR) per LV, che sono ben consolidate nella letteratura relativa alla computer vision. In particolare, dopo aver raccolto un discreto numero di testi scritti a mano in SignWriting, e data la loro natura pittorica, abbiamo studiato le possibili analogie tra la SW-OGR e le moderne tecniche di OCR per i caratteri scritti della lingua araba, indiana e cinese. Anche se tali tecniche adottano approcci simili, basati su tecniche di pattern recognition e machine learning, la loro attuazione può spesso dipendere dalla specifica notazione che deve essere riconosciuta.

Nel nostro caso, non abbiamo potuto a seguire una linea simile per la progettazione del motore SW-OGR, e abbiamo elaborato le nostre procedure di riconoscimento. La ragione di questa scelta è la particolare natura di SignWriting. Prima di tutto, a differenza di molti alfabeti, SignWriting presenta una serie di simboli che è dell'ordine delle decine di migliaia. Inoltre, poiché la scrittura a mano è di solito piuttosto imprecisa, ogni simbolo può essere disegnato in un numero (di solito dell'ordine di decine) di modi diversi, come in qualsiasi notazione manoscritta. Di conseguenza, è evidente che qualsiasi training per consentire un approccio basato sul machine learning richiederebbe una quantità irragionevole di tempo e, soprattutto, di templates. Un ulteriore motivo rende l'applicazione di tecniche di pattern recognition molto difficile, cioè la totale mancanza di norme che regolino il numero, il tipo e la posizione dei simboli all'interno di un segno scritto con SignWriting. Infatti, la libertà di composizione, che è una delle caratteristiche che rendono Signwriting uno strumento adatto per l'utilizzo quotidiano, rende anche la modellizzazione della notazione molto difficile.

Per le ragioni di cui sopra, abbiamo progettato e implementato SW-OGR in modo da eseguire il riconoscimento di testi in SignWriting lavorando esclusivamente con le caratteristiche geometriche e topologiche dei simboli, e con le loro relazioni topologiche. Abbiamo anche fatto affidamento su informazioni contestuali, come ad esempio la conoscenza dell'organizzazione dell'alfabeto di SignWriting.

In questo lavoro presentiamo i nostri studi sull'alfabeto di SignWriting al fine di identificare i criteri geometrici per classificare i suoi simboli, il design della procedura di riconoscimento, e lo sviluppo e il testing del motore SW-OGR. La parte iniziale della procedura mira ad individuare le forme base all'interno dei frammenti presenti del testo scritto (cerchi, poligoni, ecc). Questo è l'unico passo in cui possiamo sfruttare procedure di machine learning esistenti. Le forme rilevate e le loro caratteristiche sono usate per inferire la regione del corpo o il tipo di elemento non-anatomico (movi-

mento, contatto, ecc) che il frammento può descrivere. Da questo punto in poi, i criteri individuati corrispondono a una serie di controlli effettuati per riconoscere le caratteristiche del simbolo. Il risultato dei controlli vengono utilizzati per costruire un codice, che viene infine utilizzato per identificare il simbolo.

Il motore SW-OGR è destinato a servire un duplice scopo: in primo luogo, esso può essere integrato all'interno di editor di Signwriting esistenti, come SWift, al fine di fornire un supporto rapido per la scrittura a mano, e rendere il processo di composizione molto più veloce e comodo per l'uso quotidiano. Inoltre, fin dall'inizio del nostro lavoro, eravamo consapevoli della presenza (e della notevole dimensione) di un certo numero di corpus manoscritti in SignWriting, raccolti da diverse comunità in tutto il mondo. Questi corpus sono un bene prezioso, e potrebbero diventare ancora più utili se digitalizzati. SW-OGR è in grado di digitalizzare tali corpus in modo intelligente, in quanto non si limita a eseguire una semplice "scansione" dei documenti, ma, attraverso il riconoscimento, è in grado di raccogliere tutte le informazioni sul rapporto tra i segni e i simboli che li compongono, permettendo così alla comunità scientifica di eseguire qualsiasi tipo di analisi linguistica su interi dataset in SignWriting.





# Contents

<b>Introduction</b>	<b>29</b>
<b>1 Deafness and communication</b>	<b>33</b>
1.1 Deafness and language . . . . .	33
1.1.1 Sign language in history . . . . .	33
1.1.2 Sign Language and written representation . . . . .	41
1.1.2.1 Glosses . . . . .	42
1.1.2.2 The Stokoe notation . . . . .	43
1.1.2.3 The HamNoSys notation . . . . .	44
1.1.2.4 The SignWriting notation . . . . .	44
1.1.2.5 A comparison of notations for Sign Language . . . . .	45
1.2 Deafness and the digital divide . . . . .	48
<b>2 SignWriting</b>	<b>51</b>
2.1 The history of SignWriting . . . . .	52
2.2 Overview of SignWriting system . . . . .	59
2.3 A different SignWriting coding . . . . .	67
2.4 Digital SignWriting . . . . .	68
2.4.1 Digital resources in SignWriting . . . . .	68
2.4.1.1 Websites . . . . .	68
2.4.1.2 Blogs . . . . .	69
2.4.1.3 Wikipedia . . . . .	70
2.4.1.4 Digital editors . . . . .	70
2.4.2 Challenges of digital SignWriting . . . . .	71
<b>3 SWord</b>	<b>73</b>
3.1 The SWord framework . . . . .	73
3.2 SWift . . . . .	75
3.2.1 Introduction to SWift . . . . .	75
3.2.2 Core features . . . . .	78
3.2.2.1 Interface design . . . . .	79
3.2.2.2 Glyph search system . . . . .	80
3.2.2.3 Support for user-defined glyphs . . . . .	82
3.2.2.4 Toolbox design . . . . .	83
3.2.2.5 Saving options and formats . . . . .	84
3.2.2.6 Storyboard . . . . .	84

3.2.3	Usability test . . . . .	85
3.2.3.1	Methodology . . . . .	85
3.2.3.2	Results . . . . .	91
3.2.3.3	Discussion . . . . .	94
3.2.3.4	Limitations . . . . .	96
3.2.4	After the test . . . . .	96
<b>4</b>	<b>Introduction to SW-OGR</b>	<b>97</b>
4.1	Reasons to work for SW-OGR . . . . .	97
4.2	Related work on OCR . . . . .	101
4.2.1	Data acquisition . . . . .	105
4.2.2	Pre-processing . . . . .	105
4.2.3	Segmentation . . . . .	106
4.2.4	Feature extraction . . . . .	108
4.2.5	Classification . . . . .	110
4.2.6	Post-processing . . . . .	112
4.3	SW-OGR issues at a glance . . . . .	113
<b>5</b>	<b>Coding system of SW-OGR</b>	<b>115</b>
5.1	SignWriting datasets . . . . .	115
5.2	Core recognition elements . . . . .	118
5.3	OGR coding for SignWriting . . . . .	122
5.3.1	Hand Contacts (Co) . . . . .	126
5.3.2	Hand Configurations (HaC) . . . . .	132
5.4	Interoperability between coding systems . . . . .	141
<b>6</b>	<b>Implementation of SW-OGR</b>	<b>145</b>
6.1	Architecture of SW-OGR . . . . .	145
6.1.1	Concept . . . . .	145
6.1.2	Software architecture . . . . .	147
6.2	Design and development of SW-OGR . . . . .	161
6.2.1	Image pre-processing . . . . .	161
6.2.2	Image binarization . . . . .	162
6.2.2.1	Thresholding methods . . . . .	162
6.2.2.2	Thresholding performance evaluation . . . . .	163
6.2.2.2.1	Testing methodology . . . . .	165
6.2.2.2.2	Results and discussion . . . . .	167
6.2.3	Shape detection . . . . .	170
6.2.3.1	Circles . . . . .	170
6.2.3.2	Rectangles, triangles and other polygons . . . . .	175
6.2.4	OGR area inference . . . . .	176
6.2.4.1	Head Circles and Contacts (HeE_HE) . . . . .	178
6.2.4.2	Hand Configurations (HaC) . . . . .	181
6.2.5	Glyph recognition . . . . .	182
6.2.5.1	Head Circles and Contacts (HeE_HE) . . . . .	182
6.2.5.2	Hand Configurations (HaC) . . . . .	183

6.3	Development status and limitations . . . . .	193
<b>7</b>	<b>Testing of SW-OGR</b>	<b>195</b>
7.1	Testing of SW-OGR . . . . .	195
7.1.1	Head Circles and Contacts (HeE_HE) . . . . .	196
7.1.2	Facial Expressions: Eyes (HeE_EY) . . . . .	198
7.1.3	Hand Configurations (HaC) . . . . .	200
7.2	The SW-OGR Engine in action . . . . .	202
	<b>Conclusions and future research</b>	<b>205</b>
	<b>Glossary</b>	<b>217</b>
	<b>Acronyms</b>	<b>219</b>



# List of Figures

1.1	<i>Death of St. Bernardine</i> , painted by Pinturicchio in the church of St. Mary in Aracoeli. . . . .	36
1.2	Excerpt from John Bulwer’s <i>Chirologia</i> (Bulwer, 1644), presenting a very early sign dictionary. . . . .	37
1.3	ASL sign for <i>bear</i> , glossed in English. Images extracted from (Sutton, 1996). . . . .	43
1.4	ASL sign for <i>bear</i> , written using the Stokoe notation. Images extracted from (Sutton, 1996). . . . .	44
1.5	ASL sign for <i>bear</i> , written using the HamNoSys notation. Images extracted from (Sutton, 1996). . . . .	45
1.6	ASL sign for <i>bear</i> , written using the SignWriting notation. Images extracted from (Sutton, 1996). . . . .	45
1.7	ASL sign for <i>bear</i> , written using different notations. Images extracted from (Sutton, 1996). . . . .	46
1.8	A comparison between writing systems, conducted taking into account different factors. Image extracted from <i>ASL Font: Ways to write ASL</i> (2013) . . . . .	46
1.9	A group of hand configurations in use within ASL, expressed using different writing systems. Image extracted from <i>ASL Font: Ways to write ASL</i> (2013) . . . . .	48
2.1	LIS sign for <i>Fun</i> , written in SignWriting. Extracted from (Di Renzo et al., 2012) . . . . .	52
2.2	First steps of the <i>Sugar Plum Fairy</i> , from the second act of the <i>Nutcracker</i> ballet, recorded with DanceWriting (Sutton, 1983). . . . .	53
2.3	Early fragment of SignWriting (1976), extracted from (Sutton & von der Lieth, 1976). This fragment represents a Danish SL sentence meaning <i>It is father</i> . The significant influence of DanceWriting over the newborn SignWriting can be identified by the glyph shapes and by the presence of a three-lined staff. . . . .	53
2.4	Detail of the ASL Wikipedia page about Abraham Lincoln, written in SignWriting. . . . .	55
2.5	Communities which use SignWriting around the world (Sutton, 1996). . . . .	57

2.6	Difference between expressive and receptive point of view. Extracted and translated from (Di Renzo et al., 2012). . . . .	60
2.7	LIS sign for <i>Fun</i> , written in SignWriting. Different colors highlight glyphs belonging to different ISWA categories. . . . .	62
2.8	Base symbol of a hand configuration featuring a single index finger extended. Extracted from (Sutton, 1996) . . . . .	65
2.9	Orientation variations available for a glyph belonging to Category 1 - Hands. Extracted from (Sutton, 1996) . . . . .	65
2.10	Plane variations available for a glyph belonging to Category 1 - Hands. Extracted from (Sutton, 1996) . . . . .	66
2.11	Rotation variations available for a glyph belonging to Category 1 - Hands. Original configuration image extracted from (Sutton, 1996) . . . . .	66
2.12	Illustration of the 13-digit Sutton ISWA code used to uniquely identify any SignWriting glyph. . . . .	67
2.13	An example of the improvements introduced by ISWA-BIANCHINI. The white boxes show the possible trajectories of hand movements within ISWA2008, arranged by geometric planes; The orange boxes show the missing trajectories which were added to improve the system consistence. . . . .	68
2.14	Home screen of the <i>SignWriting Website</i> (available at <a href="http://www.signwriting.org/">http://www.signwriting.org/</a> ). . . . .	69
2.15	Home screen of the <i>Frost Village</i> blog (available at <a href="http://www.frostvillage.com/">http://www.frostvillage.com/</a> ). . . . .	69
2.16	Home screen of the <i>ASL Wikipedia Project</i> (available at <a href="http://ase.wikipedia.wmflabs.org/wiki/Main_Page">http://ase.wikipedia.wmflabs.org/wiki/Main_Page</a> ). A subset of the available entries are visible. . . . .	70
2.17	Screenshot of <i>SWift</i> SignWriting digital editor (available at <a href="http://visel.di.uniroma1.it/SWift/">http://visel.di.uniroma1.it/SWift/</a> ). . . . .	71
3.1	Concept illustration of the SWord framework (courtesy of Prof. Maria De Marsico). . . . .	74
3.2	The user interface of Sutton's SignMaker (Sutton, 1996). . . . .	77
3.3	SWift's (v. 1.01) home screen, divided in 4 functional areas. . . . .	79
3.4	Search interface related to the anatomic area of the hands. . . . .	81
3.5	Search interface related to the anatomic area of the hands: one criterion (glyphs involving only three extended fingers) is selected in one of the choose boxes. . . . .	82
3.6	Support for handwritten glyphs provided by SWift: the glyph is first handwritten (left) and then imported on the sign display (right). . . . .	83
3.7	Animation frames of two buttons within the interface of SWift: <i>Anti-clockwise Rotation</i> and <i>Delete</i> . . . . .	84

3.8	Storyboard: the story composition screen. The management buttons (open, save, delete) are visible in the title bar (top) and the signs already added are visible in the center of the screen, along with the buttons for their management (create, edit, delete). . . . .	85
3.9	Spatial setting for the Think by Signs Test. . . . .	87
3.10	VL representation of task number 6 of the Think by Signs Test. . . . .	89
3.11	Web page showing a question of the QUIIS employed for the test. . . . .	91
3.12	Difference between the puppet of V.1.00 (before the usability test) on the right and the puppet of V.1.01 (after the test) on the left; the circle with the hand icon highlights the mouse pointer. . . . .	96
4.1	Component diagram for a new generation of SignWriting editors featuring a SW-OGR Engine. . . . .	98
4.2	Component diagram for a new generation of mass SignWriting corpora digitalizer featuring a SW-OGR Engine. . . . .	100
4.3	Arab letters consist of strokes and dots. If they are present, dots can occur from 1 to 3 times (top). Moreover, dots can be above, in the middle or below the letter (bottom). . . . .	102
4.4	Different baseline alignments in different notations. From left to right: the alphabetic baseline (Arabic, Latin, etc.), hanging baseline (Indian scripts, etc.), ideographic baseline (East Asian ideograms, etc.). . . . .	103
5.1	Sample of the DS-PEAR dataset. . . . .	116
5.2	Sample of the DS-POITIERS dataset. . . . .	117
5.3	Sample of the DS-DEV dataset. . . . .	117
5.4	An example of text belonging to the DS-POITIERS dataset. . . . .	119
5.5	An example of slice (courtesy of C.S. Bianchini). This is the first slice extracted from the text in Fig. 5.4. . . . .	119
5.6	An example of frag (courtesy of C.S. Bianchini). This is the first frag extracted from the slice in Fig. 5.5. . . . .	120
5.7	The complex relationship between frag and glyphs: a frag may represent a set of glyphs (left), a glyph (middle) or just a part of a glyph (right - the glyph is composed by three different frags). . . . .	120
5.8	Set of frags detected within the text in Fig. 5.4. Each frag has been highlighted with a different random color. . . . .	121
5.9	A few examples of the features encoded by an OGR identifier: from left to right: presence of geometric shapes, presence of convexity defects within the convex hull, presence of foreground clusters. . . . .	122

5.10	Glyph representing the shape of an eye (ISWA-2010: 04-02-006-01-01-01). The original digital version is visible on the left, along with some examples of drawing inaccuracies, found within our datasets. . . . .	124
5.11	The whole set of glyphs within the Hand Contacts area (85 symbols). . . . .	127
5.12	Glyphs within the Hand Contacts area, without rotations (25 symbols). . . . .	128
5.13	Organization of a OGR tree. . . . .	129
5.14	Hand Contacts area after the <i>base shape</i> check. . . . .	130
5.15	Hand Contacts area after the <i>repetition</i> check. A number of internal nodes and leaf nodes are hidden for the sake of space. . . . .	131
5.16	Hand Contacts area after the <i>line</i> check. A number of internal nodes and leaf nodes are hidden for the sake of space. . . . .	131
5.17	Results of the <i>appshape</i> check, applied to different glyphs. . . . .	136
5.18	Side encoding for a glyph which features a <i>SQUARE</i> base shape. . . . .	138
5.19	Appendix encoding: <i>oncorner</i> evaluation. . . . .	139
5.20	Appendix encoding: <i>inprojection</i> evaluation. . . . .	140
5.21	Appendix encoding: <i>stangle</i> evaluation. . . . .	140
5.22	Appendix encoding: summary of all evaluations. . . . .	141
5.23	A set of glyphs belonging to the Hand Configurations area. Such symbols can be subject to drawing inaccuracy: users may draw the bottom-right appendix of each glyph either on the corner or on the side. . . . .	141
5.24	Detail of the OGR to ISWA-BIANCHINI mapping table for the Facial Expressions: Mouth area. Each ISWA-BIANCHINI code (on the left) is mapped to an OGR code (on the right). . . . .	143
6.1	High-level activity diagram describing the recognition procedure implemented by the SW-OGR Engine. . . . .	146
6.2	Package diagram of the SW-OGR Engine. . . . .	148
6.3	Class diagram of the <code>beans</code> package. . . . .	150
6.4	Class diagram of the <code>util</code> package: image utility library. . . . .	153
6.5	Class diagram of the <code>util</code> package: geometric utility library. . . . .	154
6.6	Class diagram of the <code>util</code> package: math utility library and comparison library. . . . .	155
6.7	Class diagram of the <code>util</code> package: shape classes. . . . .	156
6.8	Class diagram of the <code>model</code> package. . . . .	157
6.9	Class diagram of the <code>recog</code> package. . . . .	159
6.10	Fixed thresholding ( $t = 100$ ) applied to different slices. . . . .	163
6.11	Multiple thresholding methods applied on the same slice: test slice 1. . . . .	164
6.12	Multiple thresholding methods applied on the same slice: test slice 2. . . . .	165



---

6.13	Test series (ground truth GT1): outcome of ME and RAE metrics for image thresholding evaluation. . . . .	167
6.14	Test series (ground truth GT2): outcome of ME and RAE metrics for image thresholding evaluation. . . . .	169
6.15	Final performance results for the thresholding evaluation test.	170
6.16	A number of circles, extracted from both DS-PEAR and DS-POITIERS datasets. . . . .	171
6.17	Application of the generalized Hough transform for the detection of curves to images which contain circles. . . . .	172
6.18	Application of the generalized Hough transform for the detection of curves to images which do not contain circles. . . . .	172
6.19	Illustration of the circle detection reliability check implemented by the SW-OGR Engine. . . . .	173
6.20	Polygon detection algorithm implemented within the SW-OGR Engine, applied to glyphs featuring rectangular shapes. The algorithm uses the Shi and Tomasi method to locate the corners in each image (top). Afterwards, the corners are processed in order to build the polygon, if this is possible (bottom).	177
6.21	Two examples of signs containing glyphs belonging to the Shoulder Movements (ShM) area. The glyphs are visible right below the head circle. . . . .	177
6.22	OGR area inference check for facial expressions. A frag is inferred to represent a left eye according to the topological relationship between its MBR and the center of the head circle.	180
6.23	OGR area inference check for facial expressions. A frag is inferred to represent a left eye according to the topological relationship between its MBR and the MBR of the head circle.	181
6.24	A set of glyphs belonging to the Head Circle and Contacts area.	182
6.25	Feature recognition applied to a set of glyphs belonging to the Head Circle and Contacts area. . . . .	183
6.26	A set of glyphs belonging to the Hands Configurations area. The detected base shape is marked over each glyph. . . . .	184
6.27	A set of glyphs belonging to the Hands Configurations area, as they appear after the thinning process. The detected base shape is marked over each glyph, and the original foreground area is visible in transparency behind each glyph. . . . .	185
6.28	Start point detection of the outer appendices, applied to a number of glyphs belonging to the Hands Configurations area. The detected base shape is marked over each glyph, the scan lines are visible around the base shape, and the detected foreground points along the scan lines are marked by dots. . . . .	186
6.29	Examples of end points in a line. . . . .	187
6.30	Examples of branch points in a line. . . . .	187
6.31	Examples of cross points in a line. . . . .	187

6.32	Illustration of the initial outer appendix scan applied to a number of glyphs belonging to the Hands Configurations area. Each column of the image shows: the image of the glyph, along with its detected base shape; the POIs detected following the appendices; the detected appendices, each one highlighted with a random color. . . . .	188
6.33	Illustration of the third appendix optimization step, performed during the recognition of glyphs belonging to the Hands configuration area. . . . .	189
6.34	Illustration of the fourth appendix optimization step, performed during the recognition of glyphs belonging to the Hands configuration area. . . . .	189
6.35	Illustration of the fourth appendix optimization step, performed during the recognition of glyphs belonging to the Hands configuration area. . . . .	190
6.36	Illustration of the fourth appendix optimization step, performed during the recognition of glyphs belonging to the Hands configuration area. . . . .	191
6.37	Screenshot representing the completed recognition of a glyph belonging to the Hands Configurations area. . . . .	192
6.38	Screenshot representing the completed recognition of a glyph belonging to the Hands Configurations area. . . . .	192
7.1	A handwritten page belonging to the DS-TEST-POITIERS dataset, as it appears before (left) and after (right) the recognition of the glyphs belonging to the Head Circles and Contacts area. . . . .	197
7.2	A set of glyph belonging to the Facial Expressions: Eyes area.	198
7.3	A handwritten page belonging to the DS-TEST-POITIERS dataset, as it appears before (left) and after (right) the recognition of the glyphs belonging to the Facial Expression: Eyes area. . . . .	199
7.4	Two datasets generated to test the recognition algorithms for the Hand Configurations area. The page on the left represents a number of possible variations of a hand configuration. The page on the right represents a number of printed hand configurations. . . . .	200
7.5	A handwritten page belonging to the DS-TEST-POITIERS dataset, as it appears before (left) and after (right) the recognition of the glyphs belonging to the Hand Configurations area.	201
7.6	A handwritten page belonging to the DS-TEST-PEAR dataset, as it appears before (left) and after (right) the recognition. . . . .	203
7.7	A handwritten page belonging to the DS-TEST-POITIERS dataset, as it appears before (left) and after (right) the recognition. . . . .	204

# List of Tables

2.1	Major changes to SignWriting since 1974 to present. Information and images drawn from (Sutton, 1996). . . . .	56
2.2	Communities using SignWriting around the world, divided by continent. Information extracted from (Sutton, 1996). . . . .	58
2.3	Categories identified within the ISWA-2010. Each category is illustrated with one of its most representative glyphs. Information and images drawn from (Sutton, 1996) and (Slevinski, S.E., Jr, 2010a). . . . .	62
2.4	Group organization of Category 1: Hands, within ISWA-2010. Each group is illustrated with one of its most representative glyphs, and the relative hand configuration. Information and images drawn from (Sutton, 1996). . . . .	63
2.5	Glyph groups in ISWA-2010. . . . .	64
3.1	List of 9 tasks requested to the participants during the Think by Signs Test. . . . .	88
3.2	List of questions included in the QUIIS employed for the test. . . . .	90
3.3	Event occurrence percentage (out of the number of total events recorded) for each of the 9 tasks of the Think by Signs Test. . . . .	93
3.4	Percentage of occurrence of event sets for each task. The percentage of an event set for a given task is calculated out of the total events occurring during that given task. The most recurring event set is highlighted for each row. N/A is found only if no events were detected during a task. . . . .	93
3.5	5 most recurring events during SWift usability test. . . . .	93
3.6	Satisfaction level ( $\mu$ and $\sigma$ ) for each section of the QUIIS employed for the test. . . . .	94
5.1	Areas defined within the OGR coding system. For each area, the table reports the code, the ISWA category or group associated to the area, and the checks that SW-OGR performs, in order to check if a glyph belongs to the area. . . . .	126
5.2	Hand Contacts area - <i>base shape</i> check details. For each shape, the table reports the name, the OGR code and an example image. . . . .	129

5.3	Hand Configurations area - <i>base shape</i> check details. For each shape, the table reports the name, the OGR code, an image of the shape, and a number of example glyphs. . . . .	133
5.4	Hand Configurations area - <i>orientation</i> check details. For each orientation, the table reports the name, the OGR code and a number of example images. . . . .	134
5.5	Hand Configurations area - <i>attachment</i> check details. For each case, the table reports the name, the OGR code and a number of example images. . . . .	134
5.6	Hand Configurations area - <i>aploc</i> check details. For each case, the table reports the name, the OGR code and an example image. . . . .	135
5.7	Appendix shapes within the Hand Configurations area. For each shape, the table reports the name, a short description and a number of example images. . . . .	135
6.1	Test series (ground truth GT1): misclassification error. Minimum (i.e. optimal) values for each slice are highlighted. . . .	167
6.2	Test series (ground truth GT1): relative foreground area error. Minimum (i.e. optimal) values for each slice are highlighted.	168
6.3	Test series (ground truth GT2): misclassification error. Minimum (i.e. optimal) values for each slice are highlighted. . . .	169
6.4	Test series (ground truth GT2): relative foreground area error. Minimum (i.e. optimal) values for each slice are highlighted.	170
6.5	Ratio of the diameter of head circles to the width of their slices.	179
6.6	Development Status, updated to the 27th of August 2014. . . .	194
7.1	SW-OGR performance test: Head Circle and Contacts. . . . .	196
7.2	SW-OGR performance test: Facial Expressions: Eyes. . . . .	199
7.3	SW-OGR performance test: Hand Configurations. . . . .	201

# Introduction

The subject of the present work lies in the middle between image processing and human-computer interaction (HCI). The title reads: *Informatisation of a graphic form of Sign Languages - application to SignWriting*, and it is important to spend a few words to clarify its meaning. First of all, Sign Language (SL) is the visual-gestural language used by many deaf people to communicate with each other. SL is deeply different from the Vocal Language (VL) used by hearing people, such as English, Chinese, etc. This is mainly due to the fact that the cognitive structures underlying the language processing of deaf people are deeply different from those exploited by hearing people. In fact, such structures reflect the highly visual perception experienced by people who mainly feel the world surrounding them through their eyes.

Like many other languages in the world, SLs do not currently have a widely acknowledged *graphic* (or *written*) *form*. The main difference between the two situations is that in the former case the problem is mostly limited to small and isolated communities, while SLs, which are different in the different countries like VLs, are used by a community which shares its communication space with another (hearing) community, whose VL is the dominating one. As a matter of fact, the deaf communities around the world do not currently share a common writing system to represent their preferred language. This still happens notwithstanding the fact that a number of writing systems have been developed for SLs, by different research teams and for different purposes (see Chapter 1). Actually, choosing one writing system for SLs is subordinated to the goal that one must pursue. In our case, given our goal to increase the accessibility of electronic resources to deaf people in every day life, SignWriting proved the most appropriate candidate to work with, since it has features that can rarely be found in other SL writing systems. More specifically, SignWriting is a writing system which uses visual symbols to represent the handshapes, movements, and facial expressions of SLs. Its high iconicity, and the possibility to be employed in everyday use made it the ideal candidate for a wide diffusion, and for this work (see Chapter 2). Unfortunately, neither SLs, and even less any of the systems devised to write them, are adequately, extensively and ubiquitously exploited to transmit information in the digital world. Most digital artifacts, in fact, are available only in VL, whether they are applications, content, websites, etc. Little attention is paid to accessibility design for deaf people, which is often car-

ried out using solutions on the edge of the workaround, i.e. through textual captioning and audio-content transcription. As detailed in Chapter 1, this kind of design may support the needs of people who become deaf after the acquisition of speech and language (post-lingual deafness), but issues related to pre-lingual deafness are seldom and poorly addressed. Actually, a deaf-oriented accessibility design based on SL support is still far from being realized. Videos, more specifically *signed videos*, are typically the most widespread technique for SL inclusion. However, even if many deaf communities have recently replaced many functions of *writing* by using signed videos, this is not always possible or appropriate. As an example, it is not possible to take notes with a video, or to annotate a web resource (tagging) or to enter a query on a search engine. In other words, videos lack the ease of handling and the variegated usability of a written expression.

The belief underlying the present work is that the informatisation of SLs is of paramount importance to achieve an effective deaf-oriented accessibility design, and to ultimately mitigate the impact of the digital divide on deaf people. First of all, in order to achieve this goal, it was necessary to design a framework to support the production and the use of SignWriting-oriented resources for the deaf (SWord). SWord is a framework designed to include software for the acquisition, management and dissemination of signs and signed stories written in SignWriting. The ultimate purpose of SWord is to make SignWriting effectively exploitable as a communication mean and a learning support for deaf people, especially in the digital world (see Chapter 3). The first software to be included into the SWord framework was SWift, a digital editor which allows the creation of digital SL resources written in SignWriting. In this work, we present a number of upgraded features of SWift, such as the possibility to write signed stories. Moreover, we conducted a test session to evaluate the usability and the correct functioning of SWift; in order to do this, a sample of the main target users of SignWriting digital editors was gathered. Since most participants were deaf, the tools and the methodologies chosen for the test were adapted to work with deaf people too (see Chapter 3.2)

The HCI, however, is not the core topic of this dissertation. In fact, despite our efforts, we observed that handwriting signs is still much easier than using any SignWriting digital editor, such as SWift, to compose them. For this reason, we designed and implemented a software methodology whose purpose is to operate the electronic conversion of scanned images containing handwritten or printed SignWriting symbols into machine-encoded SignWriting texts. Such technique is referred to as the SignWriting Optical Glyph Recognition (SW-OGR) in the present work (see Chapter 4, 5, 6 and 7).

Due to the huge number of symbols to recognize (about 40.000), it was not possible to rely on Machine Learning techniques, since any training procedure aiming at recognizing the whole set of glyphs would have been unsustainably expensive. For this reason, SW-OGR performs the recognition of SignWriting texts by only working with processing rules related to the geometric and topological features of the symbols, and with their topologi-

cal relationships. We also relied on context-dependent information, such as the knowledge of the organization of the SignWriting alphabet. In this work we present the studies performed on the SignWriting alphabet in order to identify geometric criteria and rules to classify its symbols, the design of the recognition procedure, and the development and testing of the SW-OGR engine. The engine is intended to serve a twofold purpose: first of all, it can be embedded within existing SignWriting editors, such as SWift, in order to provide a prompt support for handwriting, and make the composition process much faster and comfortable for everyday use. In addition, SW-OGR is able to digitize paper SignWriting corpora in a smart way, since it does not simply perform a “scan” of the documents, but, through the recognition, it is able to gather any information about the relationship between a sign and the symbols that compose it, thus allowing the research community to perform any kind of linguistic analysis on newly-acquired SignWriting datasets. The present dissertation is composed by 7 chapters. Chapter 1 covers the necessary background about deafness and language; Chapter 2 provides a detailed description of SignWriting; Chapter 3 introduces the SWord framework and SWift. The remaining chapters cover SW-OGR. Chapter 4 introduces SW-OGR and the related work about Optical Character Recognition (OCR); Chapter 5 covers the studies on the SignWriting alphabet in order to identify the features to be exploited during the recognition procedure; Chapter 6 details the design and the implementation of SW-OGR; finally, Chapter 7 covers the testing of the application.





# Chapter 1

## Deafness and communication

### 1.1 Deafness and language

#### 1.1.1 Sign language in history

Deaf<sup>1</sup> people and the way they communicate exist since the beginning of the history of Man. Only a limited number of partial sources, mostly fragments of experiences or studies made by philosophers, poets, scientists, legislators, are available from the first centuries of documented history. Despite this, we can clearly state that the deaf were mostly ignored and, in particular times or situations, harshly opposed for a number of different reasons, ranging from pedagogical to religious (Encrevé, 2013).

Ever since the times of the Bible, prejudices against deaf people led to consider both the healing or the education of a deaf person as a miraculous event. The Holy Scriptures regarded them as incomplete and weak beings, their only chance to be part of the society was to be “touched” by Jesus.

**32** *There some people brought to him a man who was deaf and could hardly talk, and they begged Jesus to place his hand on him. 33* *After he took him aside, away from the crowd, Jesus put his fingers into the man’s ears. Then he spit and touched the man’s tongue. 34* *He looked up to heaven and with a deep sigh said to him, “Ephphatha!” (which means “Be opened!”). 35* *At this, the man’s ears were opened, his tongue was loosened and he began to speak plainly.*

(Biblica, 2011) NIV, Mark 7:32-35.

From the above citation we can notice that in those centuries (and for many to come), the causal connection between the faculty to hear sounds and the

---

<sup>1</sup>During years of work and research along with the Deaf Community, we acknowledged that the most appropriate term to refer to people with a hearing loss, is actually “deaf people”, so we will be adopting it for the present work. For further information please refer to (Cavender, Trewin, & Hanson, 2009)

faculty to produce sounds was still unclear to the most, so there was no distinction between deaf people and dumb people.

Among the early sources about deafness, we count the observations of three of the brightest minds of the ancient Greece. Plato was one of the first authors to observe and make a clear reference to a gestural language used by the deaf.

**SOCRATES:** [...] *And here I will ask you a question: Suppose that we had no voice or tongue, and wanted to communicate with one another, should we not, like the deaf and dumb, make signs with the hands and head and the rest of the body?*

**HERMOGENES:** *There would be no choice, Socrates.*

**SOCRATES:** *We should imitate the nature of the thing; the elevation of our hands to heaven would mean lightness and upwardness; heaviness and downwardness would be expressed by letting them drop to the ground; if we were describing the running of a horse, or any other animal, we should make our bodies and their gestures as like as we could to them.*

**HERMOGENES:** *I do not see that we could do anything else.*

**SOCRATES:** *We could not; for by bodily imitation only can the body ever express anything.*

(Sedley, 2003) Plato, *Cratylus*.

In his dialogue *Cratylus* (Sedley, 2003), he underlines the expressive power of a language which makes large use of the body to “mean” (σημαίνειν) anything. Notice that the whole body is observed to take part in the communication: the meaning is not exclusively conveyed by the hands but also by “the head and the rest of the body” (Sedley, 2003). Unfortunately, Plato’s findings were mostly ignored and the deaf continued to be considered like individuals without any cognitive development whatsoever. Aristoteles, and Hippocrates before him, were among the first men to reckon that speaking and hearing abilities were linked, being derived from the same area in the brain. For this reason, they believed that the deaf could never speak (Eriksson & Schmale, 1998). Their hypothesis was also supported, four centuries later, by Galen, and it was considered to be correct until the sixteenth century.

One of the few texts providing an enlightened vision about deaf people and their education is the Talmud: the first written compendium of Judaism’s Oral Law, dating back to the second century CE. In the Mishnah of the Talmud (Danby, 1933), the writers described people with disabilities as children of God who might be capable of reasoning despite their handicaps (Marschark & Spencer, 2011) and contemplate the possibility for them to be “educated and made intelligent” (Danby, 1933).

In the same age, under the Roman empire, the connection between deafness and dumbness was not understood nor accepted. This resulted in the

widespread belief that education, healing and reasoning were beyond the reach of such minority. As a consequence, Roman law (Codex Justinianus (Mommsen, Krueger, & Kroll, 2010)) specified that those born deaf had no legal rights or obligations and were forbidden to marry in addition to being required to have guardians to look after them. On the other hand, those who could read and write (mostly those who became deaf after having developed speech) were allowed to enjoy full legal rights.

Over the next millennium we find little information that might help us understand how deaf people lived. It seems likely, however, that the Middle Ages were especially dark for deaf people. It is not possible to identify a consistent vision or attitude towards deaf people, in this period. On the one hand, deaf people were often denied to enter churches, due to improper or extreme interpretations of the Holy Scriptures (see the citation below), which led people to think that hearing was a necessary condition to receive the “Word” of God.

*17 Consequently, faith comes from hearing the message, and the message is heard through the word about Christ.*

(Biblica, 2011) NIV, Romans 10:17.

On the other hand, they were well accepted in communities observing the monastic silence, such as the Benedictine Monks. On the one hand, deaf people were integrated into the agricultural and artisan society of the Middle Ages, since their hearing loss did not affect their ability to work. On the other hand, they were deemed useless by the feudal lords, due to their inability to receive orders in battle (and, as a consequence, to fight). To summarize this first period, we could define it as the “prehistoric age” of deaf history, an age of prejudice and discrimination, which often saw the deaf and the fool treated alike, without any education or legal representation.

A more complex view of deaf people and their education was developed under the Renaissance, in conjunction with the major changes in creative thinking brought by this period. In the late thirteenth century, the Dutch humanist Rudolphus Agricola described a deaf person who had been taught to read and write (Marschark & Spencer, 2011). With signs, he explained, or some other visual or pedagogical means, deaf people could sufficiently express themselves and understand the world (Radutzky, 1993). Due to the slow diffusion of literacy among the masses, it was still very unusual to find deaf people who could read and write. Even so, records identified some notable deaf artist leading productive lives. As an example, Bernardino di Betto Biagi, a.k.a. Pinturicchio, born in 1454, painted many frescoes of capital importance, among which are those about Moses’ life in the Sistine Chapel, those in Borgia Apartments in the Vatican and the *Death of St. Bernardine* in the church of St. Mary in Aracoeli (Fig. 1.1).



*Figure 1.1: Death of St. Bernardine*, painted by Pinturicchio in the church of St. Mary in Aracoeli.

Juan Fernàndes de Navarrete, a.k.a. El Mudo, a painter of Philip II of Spain, was best known for his exquisite coloring and experimentation with light. Deafened in 1529 at the age of three, he communicated in signs with the curate of the parish of Santo Vincente who found them “as intellegible as speech” (Lang & Meath-Lang, 1995). Navarrete died in 1579, and on his death bed, with pen and paper, he write his own will and appointed an executor (Marschark & Spencer, 2011).

In the same years, in San Salvador Monastery in Oña, the milestone teaching work of the Spanish Benedictine monk Pedro Ponce de Leon was developing. His deaf students were almost all children of wealthy aristocrats who could afford private tutoring, such as Francisco and Pedro de Velasco: the deaf sons of the Constable of Castille Juan Fernández de Velasco. The work of Ponce de Leon focused on helping them to learn how to speak, read and write Vocal Language (VL), Spanish in particular. He also taught his students the basics of mathematics, the principles of Catholic Religion, ancient and foreign languages, natural philosophy and politics. Unfortunately, the writings of the Benedictine monk have been almost completely lost, so we do not know whether he developed a working Sign Language (SL) or not, nor we can directly evaluate the success of his teaching method. Nevertheless, thanks to the witness of Pedro de Tovar, one of his students, we know that Ponce made large use of the Franciscan dactylological alphabet (Encrevé, 2013), which would allow a student who mastered it to spell out, letter by letter, any word. The success of Ponce’s methods is indirectly reflected by the success of his students as individuals, thus witnessing the fact that deaf people had found ways to communicate in Renaissance Europe. These appear to be the first indications of the empowerment of deaf people through education (Marschark & Spencer, 2011).

The next two centuries (seventeenth and eighteenth) brought an increase in the academic attention towards deaf people but also laid the foundation for one of the most notable philosophical conflicts in the history of the education of deaf learners: the controversy over the use of signed and spoken communication methods.

The English natural philosopher John Bulwer published, in the late seventeenth century, a number of works about deaf education, illustrating the use of natural language and (hand) gestures. His most notable works were:

- *Chirolugia: or the naturall language of the hand* (Fig. 1.2) was a compendium of manual gestures, citing their meaning and use from a wide range of sources, mainly religious and medical. *Chirolugia* was published in London in 1644, in a single volume along with *Chironomia: or, the art of manuell rhetoricke* which was a manual for the effective use of gesture in public speaking.
- With *Philocophus: or, the deafe and dumbe mans friend*, published in London in 1648; with this work, Bulwer became the first person in Great Britain to discuss the possibility of educating deaf people. To persuade the British “rational men” of “the philosophical verity of this Art” (the education of deaf people), in this volume he explained the theory and empirical evidence for its possibility.



Figure 1.2: Excerpt from John Bulwer's *Chirolugia* (Bulwer, 1644), presenting a very early sign dictionary.

On the other hand, the Swiss physician Johann Konrad Ammann and his followers focused mainly on oral communication method. In his 1693 work *The Talking Deaf, or a method proposed whereby he who is born deaf may learn to*

*speak*, the voice is regarded as the primary means of communicating human language since it alone is “the expressive secret of the soul” (Ree, 1999). Ammann’s teaching method consisted principally in focusing the attention of his deaf pupils to the motions of his lips and larynx while he spoke, and then inducing them to imitate these movements, until he brought them to repeat distinctly letters, syllables and words. Thanks to his achievements, Ammann is regarded as one of the fathers of modern Speech and Language Therapy.

One myth, perpetuated even into modern times, was the belief that abstractions could not be conveyed through SL. Yet, the anecdotes of this early period (Marschark & Spencer, 2011) reveal that the signs used by deaf people contradicted this view. Public schools were not yet established, and we have little information about how deaf children were taught individually, but we do know there were communicative exchanges between hearing people and partially, if not fully educated, deaf people.

During the Age of Reason, as the scientific societies grew in Europe, scientists and philosophers expanded their interests. Jean-Jacques Rousseau, instigator of the French Revolution, and his countryman Denis Diderot were among the most notable people to examine the potential of deaf youth to learn. Rousseau was an early influential proponent of “learning by doing”. In his book *Émile*, he expressed views which became the basis for the new order established in France after the Revolution. He redirected attention to learning through the senses and to the importance of the child’s interaction with the environment, rather than through rote memorization of the classics. As a member of the French Academy of Sciences, Rousseau took a special interest in examining deaf children instructed by a teacher named Jacob Péréire, who was using pronunciation, fingerspelling, and lipreading. As a result of the work of Rousseau and others, the instruction of deaf pupils gained increasing respect as a profession (Marschark & Spencer, 2011).

Another milestone, on the road of deaf self-determination (and SL development), was set during the 1760s: under the guidance of Charles Michel de l’Épée, France established the world’s first government-sponsored school for deaf children: the National Institution for Deaf-Mutes in Paris. Using his solid background of language theory exposed by Locke, Diderot, Condillac, Rousseau and others, the Abbé de l’Épée was able to see language as more than a verbal system of sounds. He believed that SL was the most natural way for deaf people to communicate, and, through a combination of signs and written characters, he believed it was possible to teach deaf students to think logically. The method developed by De l’Épée, however, had a significant drawback, since the SL he adopted for teaching was shaped to reproduce the syntax of French VL. As a consequence, the language used by the Abbé was rather a signed version of the French VL, rather than a proper SL. Besides the obvious consequence of producing an altered SL, such a language sometimes proved very cumbersome and hard to understand, to the extent that Jean-René Presneau speaks of “rébus linguistique et gestuel” (Encrevé, 2013; Presneau, 1998).

The clash between oralist and manualism, saw the Abbé de l'Épée and his successor, Roch Ambroise Sicard, particularly assailed by Samuel Heinicke, who established a school in Leipzig in 1778 based on the practice of teaching deaf pupils to speak. Influenced by the writings of Amman, Heinicke was one of the first to try to link speech to higher mental processes, arguing that articulation and vocal language were necessary for abstract thought (Lane, 1984). The European founders of manualism (the Abbé de l'Épée) and oralism (Heinicke) exchanged letters expressing their irreconcilable differences on educating deaf students. The “war of methods” was thus born between the proponents of the systematic use of SL in educating deaf children and those who stressed the use of speech, lipreading, and residual hearing without signs as an all-encompassing solution.

Among the students of Abbé Sicard, in the National Institution for Deaf-Mutes in Paris, there was Louis Laurent Marie Clerc, which became later known as “The Apostle of the Deaf in America” by generations of American deaf people. Clerc was a 30-year-old assistant teacher when he met Thomas Hopkins Gallaudet, which was learning the “french” method of instruction for deaf children in the same, world-famous school. Gallaudet was able to convince Clerc to accompany him to Hartford, Connecticut, where they obtained funds to establish the Connecticut Asylum for the Deaf and Dumb (now named the American School for the Deaf) in 1817. Gallaudet was its director, and Clerc became the first deaf teacher in America.

After Clerc, tens of deaf people played instrumental roles in founding educational institutions in the United States. Some became superintendents. By 1850 there were more than fifteen residential schools serving deaf pupils, with nearly 4 out of every 10 teachers in these schools deaf themselves. It was not long before that proposals for high schools and “high classes” for deaf pupils were presented at national conventions and published in journals for educators. Higher education for deaf people received a great impetus in 1857 when Amos Kendall, the business manager for Samuel F. B. Morse and his telegraph business, met with Edward Miner Gallaudet, the son of Thomas Hopkins Gallaudet, and encouraged him to accept the responsibility as the superintendent of a school for deaf and blind children which Kendall had established the previous year in the District of Columbia. The Columbia Institution for the Deaf, Dumb and Blind, incorporated by the United States Congress that year, was authorized to grant college degrees in the liberal arts and sciences. Years later, the college would become Gallaudet College and later Gallaudet University (Marschark & Spencer, 2011).

The 1880 Congress of Milan, however, resulted in an explicit denial of the emerging deaf empowerment. Congress participants, overwhelmingly hearing educators, voted to proclaim that the German oral method should be the official method used in schools of many nations.

*“The congress, considering the incontestable superiority of speech over signs, for restoring deaf-mutes to social life and for giving them greater facility in language, declares that the method of*

*articulation should have preference over that of signs in the instruction and education of the deaf and dumb*"

(quoted in Lane (Lane, 1984), p. 394)

The congress was properly orchestrated by the oralist faction: many of the proponents of SL communication were unable to attend, and deaf people were excluded from the vote. Deaf communities around the world raged at what they saw as the oppressive strategies of the hearing authorities in the schools. Actually, the outcome of the congress mainly affected France, since Italy and Germany had already adopted the oralist method before 1880, while United States remained a hard terrain for oralism even after the congress. In fact, as a result of the Milan vote, the National Association of the Deaf (NAD) was established in the United States to strengthen the political clout of deaf people, who wanted to have control over their own destiny. The choice of communication methods was a human rights issue, in reality, and one that remains volatile today (Marschark & Spencer, 2011). Despite the controversy, educators of the late nineteenth and early twentieth centuries established a rich knowledge base, publishing their perspectives on teaching in the *American Annals of the Deaf and Dumb*, which began in 1847. The cultural perspective was bolstered particularly by the scientific recognition of American Sign Language (ASL) as a true language. Through the publication of his work (Stokoe, 1960), William Stokoe proved instrumental in changing the perception of ASL from that of a broken or simplified version of English to that of a complex and thriving natural language in its own right, with an independent syntax and grammar, as functional and powerful as those underlying VLS. This led to ASL receiving more respect and attention in academic and educational environments. Stokoe also invented a written notation for SL (now called Stokoe notation) as there was no written form at the time.

Greater public awareness and acceptance of ASL was accompanied by a growing political voice among people who were deaf and hard of hearing. The social and political transformations that took place led to wholly new lifestyles for many deaf people in America as well as improved attitudes about deafness in general.

Finally, another important breakthrough in the study of SLs was made by Christian Cuxac (Cuxac, 1996, 2000) during the last decade of the twentieth century. Gathering substantial empirical evidence during his studies on the French Sign Language (LSF), Cuxac demonstrated that the face and its components, the gaze above all, were to be considered as production parameters of SL as much as (and in many cases more than) the hands. Cuxac also observed the existence of two important classes of signs within the LSF, such classes have been later observed in Italian Sign Language (LIS) and other SLs as well (Antinoro Pizzuto, 2009). The first class is composed by standard lexicon of SLs, the so called *standard signs* (Cuxac, 1996). In other words the first class is composed by the signs that are listed within SL dictionaries, such as *cat*, *house*, etc. However, SLs can also take advantage of another



way to refer to people, items, events, etc. In fact, SL users can produce signs which cannot be listed as standard signs, since they are distinguished at a meta-linguistic level (Antinoro Pizzuto, 2009) by a very particular use of the gaze, presenting highly iconic features. As an example, one of the most common iconic features is the *transfer of person*. During a SL production (i.e. a signed speech), the whole body of the signer reproduces one or more actions taken by (or occurred to) one or more characters within the production. The narrator “becomes” the person he/she is talking about, assuming the look of the represented entity, and assuming facial expressions, body postures and hand configurations which are iconically congruent with the action or state of the represented entity (Antinoro Pizzuto, 2009). Due to their highly iconic content, according to the model proposed by Cuxac, such signs are referred to as *Highly Iconic Structures* (Cuxac, 2000).

Please notice that the purpose of this section is to give a global idea of the history of SLs. For this reason, we decided to enumerate only the major events, characters and studies, since providing full-depth details about the rich linguistic and historical literature about SL is far beyond the scope of the present work.

### 1.1.2 Sign Language and written representation

According to (Antinoro Pizzuto, Rossini, & Russo, 2006), “for all the language communities that use it, a writing system is a socially shared code employed for the transmission of texts, overcoming time and space limitations”. Different types of “communicative needs”, can be at the origin of a writing system (e.g. the possibility of fixing and transmitting a shared corpus of laws, the transmission of literary texts, the elaboration of written dictionaries and grammars for educational purposes). Therefore, writing system are created in order to respond to communicative, artistic and educational needs and are designed for that (Antinoro Pizzuto et al., 2006).

Writing systems undoubtedly provide an analysis of language structures which must be sufficient to achieve particular goals, but it is typically not an exhaustive analysis of the language structures. The linguistic structures which are codified in a writing system are the ones necessary to vehiculate the meanings which are communicated in particular settings and for particular purposes or usage. Therefore, “different societies and cultures (e.g. the Chinese written culture vs. the western tradition) choose different aspects of a language in order to better achieve these ends. This always occurs through a social process of elaboration, diffusion and institutionalization of the writing system” (Antinoro Pizzuto et al., 2006).

Moreover, writing systems contribute to the standardization processes and thus influence linguistic norms, providing structures that are to be conceived as a model of a socially approved, “well formed” way of using a language (Antinoro Pizzuto et al., 2006).

The specificity of SLs is that they currently do not have a writing system. This makes them comparable to languages which feature an oral tradition

only, which currently constitute the majority of the existing languages (Ong, 2002). Since many VLS and the majority of SLs do not have their own writing system, they are often written using the writing system of other languages. In the specific case of SLs, this carries many issues and complications. Above all, the fact that during face to face communication, SL users express their thoughts in a multilinear way, by vehiculating information through the gaze, hands, facial expression, head and shoulders. More specifically, the structure of SLs is deeply different from the linear, sequential organization of VLS, due the presence of a 4-dimensional structure (spatial plus temporal) and even more due to possible overlapping of SL components (both manual and non-manual) along each dimension. Another important aspect that needs to be considered is that the relation between the concept of sign (in SL) and the concept of word (in VL) is very complex and one. In fact, a single sign may not merely express a single word; signs are more often equivalent to complex VL expressions (a concept, a situation, etc.). For more information about this aspect, please refer to Antinoro Pizzuto and Pietrandrea (2001); Antinoro Pizzuto et al. (2006). Besides the mentioned multilinearity of such languages, and the complex relationship between signs and words, we have also to consider the key role of non-manual components, of the iconic and spatial components, and even the particularity of the channel used (visual-gestural). The presence of a multilinear organization has never been observed in any VL (Garcia & Sallandre, 2013), as a consequence any writing system “borrowed” from VLS have a very limited applicability for SLs. One of the major examples is the International Phonetic Alphabet (IPA), which can be used to transcribe any VL in the world, but proves totally useless to represent SLs.

This and other difficulties (Antinoro Pizzuto et al., 2006) have undermined the development process of a written form for SLs so far. The development process is also hindered by the fact that “deaf signers live in a diglossic environment, in which their unwritten face-to-face SL must co-exist with the dominant spoken and written language used by, and in interaction with, the surrounding hearing community” (Di Renzo, Lamano, Lucioli, Pennacchi, & Ponzio, 2006). However, due to well known difficulties engendered by deafness, most deaf signers, including several highly skilled and qualified deaf researchers, do not develop appropriate literacy skills in the dominant written language (Garcia, 2006).

Despite the above difficulties, a number of writing system have been devised for SLs. However, most of them were not designed for every day, wide use, and none of them has been accepted by the Deaf Communities worldwide as the main writing system for SL (see Channon and van der Hulst (2010) for further information). The following sections (Section 1.1.2.1, 1.1.2.2, 1.1.2.3, 1.1.2.4) present some of the most notable and widespread writing systems for SLs in the world, and Section 1.1.2.5 covers a comparison of such systems.

### 1.1.2.1 Glosses

From the very beginning of linguistic research related to SLs, researchers have resorted to graphic representations based on *glosses*, i.e. the representation of isolated SL signs (or sequences) by written words (or sequences) of the national VL (Garcia & Sallandre, 2013). The word chosen to represent each sign is what might be considered the “name” of the sign, and typically represents one meaning of the sign, though not necessarily the meaning of the sign in context (*ASL Font: Ways to write ASL*, 2013). Additional information about the way each sign is produced (including specific direction, motion, repetition, and non-manual features) is indicated using a few standard symbols.

The main problem of the glosses is that the reader cannot by any mean reconstruct SL production that was there in the first place. “There is no independent representation of the signs. Everything we have is simply a sequence of textual labels that were assigned to forms that are just not there.” (Antinoro Pizzuto et al., 2006).

Questions about the inadequacy of VL glosses for SLs are thus unavoidable. Ultimately, for Antinoro Pizzuto et al. (2006) and Garcia (2006) such problems are epistemological. Antinoro Pizzuto and Pietrandrea (2001) have stressed that the so called gloss of SL cannot claim the status of writing system in the sense the term has in spoken language research, to the extent that it lacks the initial level of transcribing the signifier form. This is not a problem in the annotation of any VLs, even those without their own writing system, since spoken languages can always be phonetically transcribed using the IPA.

Fig. 1.3 shows the ASL sign for *bear*, glossed in English. Please notice that the glossed notation (on the right) does not preserve anything to reconstruct the original sign (on the left). In other words, even if the sign for *bear* was totally different from the one shown in Fig. 1.3, we would still have the very same gloss.



Figure 1.3: ASL sign for *bear*, glossed in English. Images extracted from (Sutton, 1996).

### 1.1.2.2 The Stokoe notation

W. Stokoe proposed his notation in 1960, as a part of the demonstration of the linguistic status of ASL (Stokoe, 1960). In fact, the characters of the Stokoe notation, i.e. *cherems*, are intended to be equivalent to phonemes in VLs. The system is composed by 55 cherems (19 handshapes, 12 locations, 24 movements). A large number of characters representing the cherems was

borrowed from the Latin alphabet and the numerical system, while others were invented anew for the purpose of writing SL. The notation is arranged linearly on the page and it is generally devoid of iconicity. Each sign is written following a strict order, specifying its location (first), handshape (second), and movement (third).

The Stokoe notation has a number of important limitations: first of all, the system does not cover facial expressions, mouthing, eye gaze, and body posture, as Stokoe had not worked out their phonemics in ASL (Kyle, Woll, & Pullen, 1988). There is also no provision for representing the relationship between signs in their natural context, which restricts the usefulness of the notation to the lexical or dictionary level. Moreover, the notation is only restricted to the symbols needed to meet the requirements of ASL rather than accommodating all possible signs in any SL. Finally, the Stokoe notation was never designed for everyday use, in fact, its complexity make it mostly intended for linguists and academics.

Nonetheless, Stokoe demonstrated for the first time that a SL can be written phonemically just like any other language. As a consequence, the model of the Stokoe notation was the direct source of the vast majority of systems used over the next two decades (1960 - 1980), as linguistic studies of other sign languages developed (Garcia & Sallandre, 2013).

Fig. 1.4 shows the ASL sign for *bear*, written using the Stokoe notation. A complete description of the writing system is available in Stokoe (1960)



Figure 1.4: ASL sign for *bear*, written using the Stokoe notation. Images extracted from (Sutton, 1996).

### 1.1.2.3 The HamNoSys notation

One noteworthy system based on Stokoe's is the Hamburg Notation System (HamNoSys) (Prillwitz, Leven, Zienert, Hanke, & Henning, 1989). HamNoSys is intended to enable the phonetic transcription of all SLs, and therefore includes a considerable number of symbols (more than 200 basic symbols). Since 1985, the system was gradually enhanced for the notation of spatial cues and non-manual aspects (facial expressions, body movements, eye gaze). Like the Stokoe notation, HamNoSys is arranged linearly on the page, and each sign is written following a strict order, specifying its handshape (first), orientation (second), location (third), actions (fourth).

Unlike Stokoe's system, however, "HamNoSys employs iconic symbols and shows strong internal systematicity. Yet, it faces a serious legibility problem, particularly for the recording of discourse" (Garcia & Sallandre, 2013). Nev-

ertheless, thanks to its fast digitization and compatibility with annotation software (see 3.2 below), it is integrated in large lexical databases (Hanke & Storz, 2006).

Since HamNoSys can often prove quite cumbersome to read and write, it is not employed for everyday use, but it has been frequently used in SL research. Fig. 1.5 shows the ASL sign for *bear*, written using the HamNoSys notation. A complete description of the writing system is available in Prillwitz et al. (1989)



Figure 1.5: ASL sign for *bear*, written using the HamNoSys notation. Images extracted from (Sutton, 1996).

#### 1.1.2.4 The SignWriting notation

Currently, the only system designed with the twofold objective of serving both for research and communication is SignWriting (Sutton, 1977, 1995), which can currently claim to approach the status of writing system (Garcia & Sallandre, 2013).

SignWriting is designed to be able to write any SL precisely, and uses many symbols, including symbols for writing non-manual features. It is written top to bottom in columns. The symbols can be rotated in 8 directions (in most cases) and placed anywhere in the writing area (always). Symbols indicating location, handshape, and movement are mixed together and arranged to create a “picture” of the sign.

Although it is an alphabetic type of notation, like its predecessors, “the significant innovation in SignWriting lies in its semiographic aspect, adding the analogical to the digital” (Garcia & Sallandre, 2013). In fact, the system represents all gesture production as a multi-parameter composition and as a whole (each “graphic cell” includes, analogically, the symbols of various articulators, allowing the reader to see a body and a gaze within a space), thus allowing a detailed reconstruction of spatial phenomena (Garcia & Sallandre, 2013).

Fig. 1.6 shows the ASL sign for *bear*, written using the SignWriting notation. A complete description of the writing system is available in Sutton (1977, 1995, 1996). Since SignWriting plays a very important role in the present work, a detailed description of its features is covered in Chapter 2.



Figure 1.6: ASL sign for *bear*, written using the SignWriting notation. Images extracted from (Sutton, 1996).

### 1.1.2.5 A comparison of notations for Sign Language

The comparison between writing systems for SL is a complex linguistic matter that lies well beyond the borders of computer science. The purpose of the present section is to provide a qualitative and visual comparison of the notations which were just introduced, and to explain the rationale behind our decision to choose SignWriting as the writing system to focus on. For a more detailed comparison between writing systems for SLs, please refer to Channon and van der Hulst (2010); Garcia and Sallandre (2013).

It is worth starting with our running example of the ASL sign for *bear*, showing all the introduced notations in one single picture. Observing the different ways of representing the sign, it is possible to notice that the glossed notation carries no information whatsoever about the original sign. Moreover, it is possible to notice the higher level of iconicity of SignWriting with respect to the other notations. In fact, looking at the SignWriting representation, it is possible, even without knowing the system, to identify a head and two hands, and to notice that their spatial arrangement is completely consistent with the original (face to face) sign.

A simple qualitative comparison can be performed by evaluating the writ-

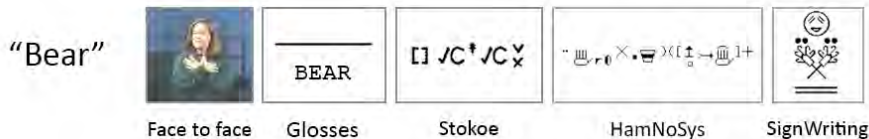


Figure 1.7: ASL sign for *bear*, written using different notations. Images extracted from (Sutton, 1996).

ing systems according to a number of criteria. The evaluation is intended to provide a high-level view of the features of the different systems. Such coarse-grained evaluation can prove particularly useful, especially when working for the informatisation of the written form of SLs, which is exactly our case. The comparison is reported in Fig. 1.8, and it has been extracted from *ASL Font: Ways to write ASL* (2013). For each representation system, the following features have been evaluated:

- Whether it can be used to write any SL in the world, or, for any reason, its usage is restricted to ASL.

Writing System	ASL-only	Non-manual grammar support	Intent	Arrangement	Computer-friendliness
<u>Gloss</u>	No	Yes	Academic	Linear	ASCII
<u>Stokoe</u>	Yes	No	Dictionary / Academic	Linear	Custom Font, or ASCII
<u>HamNoSys</u>	No	Some	Academic	Linear	Unicode PUA, Custom Font
<u>SignWriting</u>	No	Yes	Public Use	Pictorial	ASCII (machine readable), Unicode (proposed)

Figure 1.8: A comparison between writing systems, conducted taking into account different factors. Image extracted from *ASL Font: Ways to write ASL* (2013)

- Whether it can be used to write non-manual components (facial expression, body movements, prosody, eye gaze) or not.
- What is the purpose that the writing system serves (academic, everyday use, etc.).
- The graphical arrangement of the characters of the writing system.
- The character set allowing the writing system to be used in the digital world.

As explained in the previous sections, we can observe in Fig. 1.8 that the Stokoe notation does not cover non-manual components, and it cannot be used to write any SL different from ASL. HamNoSys provides more features, but its use is limited to the academic context. SignWriting, on the other hand, is the only writing system to provide a pictorial organization, and can be effectively employed for everyday use. Glosses have been included in such comparison, but they cannot definitely be considered a writing system for SL, since their features make them only useful as an annotation system for SLs (Antinoro Pizzuto & Pietrandrea, 2001; Antinoro Pizzuto et al., 2006). Another interesting visual comparison (*ASL Font: Ways to write ASL*, 2013) can be performed by evaluating the iconicity of different writing systems. We decided to focus on the handshape, since it is one of the most important components that make up a sign in any SL. The visual comparison is provided by expressing a number of hand configurations using each writing system introduced in the previous sections. More specifically, Fig. 1.9 shows a number of configurations which are in use within the ASL, expressed using different writing systems. Observing Fig. 1.9, it is possible to identify a clear difference between systems with little or no iconicity (Stokoe notation and glosses), and high iconicity (SignWriting).

Of course, the choice of a writing system is subordinated to the goal that one must pursue. The present work is focused on the informatisation of the written form of SLs, and the ultimate goal of our efforts is to contribute in leveling the digital divide affecting deaf people, and SL users in general.

As a consequence, SignWriting proved the most appropriate candidate to work with, since it has features that can rarely be found even in modern SL writing systems. Its high iconicity, and the possibility to be employed in everyday use made it the ideal candidate for a wide diffusion.

Handshape	Glosses	Stokoe	HamNoSys	Sign Writing
	1	G		
	D	G		
	L	L		
	Bent-X	X		
	X	X		
	Bent-L	X`		
	Open-G Small-C	L"		
	Small-O	L#		
	T	T		
	G	G		
	Closed-G	G#		
	F	F		

Figure 1.9: A group of hand configurations in use within ASL, expressed using different writing systems. Image extracted from *ASL Font: Ways to write ASL* (2013)

## 1.2 Deafness and the digital divide

According to the World Health Organization (World Health Organization, 2012), 278 million people worldwide are deaf or have hearing difficulties. Such people also share the same issues when approaching the digital world, since they must generally overcome barriers which are very similar to those they face in everyday life.

This is mainly due to the fact that the languages they use, i.e. SLs, are “a set of linguistic minorities relatively underrepresented in the digital world. Thus, members of the deaf community usually are confronted with websites



not written in their native language, which may arise accessibility barriers” (Fajardo, Vigo, & Salmerón, 2009). As a matter of fact, deaf people are among those groups of individuals with special needs who are most heavily affected by the digital divide. Despite the increasing attention towards accessibility issues, inclusion design for deaf people is often carried out using solutions on the edge of the workaround. In fact, most accessibility guidelines that should address their needs just deal with textual captioning and audio-content transcription (Fajardo et al., 2009). The absence of complete, accurate and universally recognized guidelines does not contribute to overcome this digital divide between deaf and hearing people. Moreover, this way of approaching the problem reveals an old misunderstanding: as explained in Section 1.1, the cognitive structures underlying the language processing of deaf people are deeply different from those exploited by people experiencing VL since infancy. Among the possible reasons for this peculiarity, deaf people base their perception and cognitive structuring of information from exterior world mostly on the sense of sight. Therefore, they tend to reflect their visual organization of the world over the organization of language. Given this peculiarity, most deaf people find significant difficulties in both learning and mastering VLs, even in their written form (Perfetti & Sandak, 2000). Even if they succeed in mastering a VL, dealing with VL content may prove quite a tiring task for them, unless it is performed for a short time (Antinoro Pizzuto, Bianchini, Capuano, Gianfreda, & Rossini, 2010). In fact, it can be observed that most of them prefer to communicate using SLs (Antinoro Pizzuto et al., 2010). Therefore, dealing with the issues of deaf-oriented accessibility using written VL is quite unrealistic (Borgia, Bianchini, & De Marsico, 2014), and any VL-based solution (VL captioning and transcription) to overcome the digital divide rarely solves the problem. In summary, while captioning-based accessibility design may support the needs of people who become deaf after the acquisition of speech and language (post-lingual deafness), issues related to pre-lingual deafness are seldom and poorly addressed. Despite some deaf people can use VL, dealing with the issues of deaf-oriented accessibility using written VL is quite unrealistic (Borgia et al., 2014).

The Web Content Accessibility Guidelines (WCAG) document, produced by the World Wide Web Consortium (W3C) is currently one of the few attempts to address general accessibility issues, including deaf-related ones. However, this document mainly addresses blindness, while there is little understanding of deaf people’s problems (see Perfetti and Sandak (2000) for a discussion). In particular, in the first version WCAG 1.0 (Vanderheiden, Chisholm, & Jacobs, 1999), the deaf-centered guidelines are quite vague and they just deal with labeling and audio-content transcription, leaving SL visual-spatial features unmanaged. WCAG 2.0 (Caldwell, B. and Cooper, M. and Guarino Reid, L. and Vanderheiden, G., 2008) addresses such issues in a better way. For instance, the success criterion 1.2.6, whose satisfaction is mandatory to achieve the highest level of compliance (AAA), states that “Sign language interpretation is provided for all prerecorded audio content

in the form of synchronous media types”.

Despite the efforts made by the W3C, a widespread support for SL in the digital world is still far from being realized. Some research projects have developed a series of techniques to allow deaf people to access digital information through different forms of deaf-oriented hyperlinking. A seemingly simple idea is the basis of the Cogniweb project (Fajardo, Parra, Cañas, Abascal, & Lopez, 2008), which tested two alternatives to feature SL videos intended to support navigation, which are embedded within small frames. In both cases, each video contains a SL translation (produced by a signing person) of any text hyperlink. This approach was called Sign Language Scent, as an alternative to provide Information Scent (Pirolli & Card, 1999), which is usually embedded in well-designed textual links. In the first proposed technique, the video frame for to this task is located at the bottom of the page, and starts the corresponding SL sequence as the user hovers the mouse over a link. In the second technique, an embedded (mouseover-activated) video is set near each link. Two tests demonstrated that deaf people can navigate more efficiently using the second technique. Other, more advanced approaches are aimed at producing digital content (more specifically, web content) using exclusively SL. Those approaches are based on Hypervideo technology, which features hyperlinks embedded within a video, allowing to retrieve further information about the concepts expressed in the video. The SignLinking system (Fels, Richards, Hardman, & Lee, 2006), based on Hypervideo, has a similar, yet more advanced interaction modality than Sign Language Scent. Each SignLink spans a time window within the video, where the presence of a link is usually indicated by an *ad hoc* icon. As the icon appears, the user can choose whether to follow the link or to keep watching the video. It is still unclear which method, among the described ones, is best suited for SL navigation (Fajardo et al., 2009).

Though effective and technologically smart and attractive, the above and similar current approaches to support digital content for deaf people share the same flaws. Even if the deaf community has recently replaced many functions of *writing* by videos, this is not always possible or appropriate. As an example, it is not possible to take notes with a video, or to annotate a web resource (tagging) or to enter a query on a search engine. In other words, videos lack the ease of handling and the variegated usability of a written expression.

Concluding the present chapter, it is important to remark that we strongly believe that including support for SL, within digital artifacts (whether they are software applications, websites, etc.) is necessary in order to address accessibility issues for deaf people, and ultimately to overcome the digital divide affecting them. Such belief is shared with many researchers and supported by scientific evidence, as shown in the this section.

The present work continues with the description of SignWriting: the writing system for SL that we adopted to perform our studies and to produce our software applications.

## Chapter 2

# SignWriting

*SignWriting is a writing system which uses visual symbols to represent the handshapes, movements, and facial expressions of signed languages. It is an “alphabet” - a list of symbols used to write any signed language in the world.*

*The SignWriting alphabet can be compared to the alphabet we use to write English, the Roman alphabet. The Roman alphabet can be used to write many different spoken languages. While each language may add or subtract one or two symbols, the same basic symbols we use to write English are used to write Danish, German, French and Spanish. The Roman alphabet is international, but the languages it writes are not.*

*In the same way, the symbols in the SignWriting alphabet are international and can be used to write American Sign Language, Danish Sign Language, Norwegian Sign Language, British Sign Language, Dutch Sign Language - any signed language you choose.*

(Sutton, 1996) Valerie Sutton - What is SignWriting

SignWriting (Sutton, 1977, 1995) is a featural<sup>1</sup> and iconic writing system for SLs. In SignWriting, a combination of 2-dimensional symbols, named *glyphs* is used to represent any possible sign in SL. The glyphs are abstract images depicting positions or movements of hands, face and body. Fig 2.1 shows the LIS sign for *Fun*, written in SignWriting.

The high iconicity of this system is due to the shapes of the glyphs themselves, which are abstract images depicting positions or movements of hands, face, and body. SLs are characterized by a 3-dimensional spatial arrange-

---

<sup>1</sup>A featural alphabet is an alphabet wherein the shapes of the letters are not arbitrary, but encode phonological features of the phonemes they represent. The term featural was introduced by Geoffrey Sampson to describe Hangul (Sampson, 1990).

ment of gestures and by their temporal structure, In the very same way, the spatial arrangement of the glyphs in the page plays a core role, since it does not follow a sequential order (like the letters of the written form of VLs) but it follows the natural arrangement suggested by the human body. Since SignWriting, as a featural system, it represents the actual physical formation of signs rather than their meaning, no phonemic or semantic analysis of a language is required to write it. A person who has learned the system can “feel out” an unfamiliar sign in the same way an English speaking person can “sound out” an unfamiliar word written in the Latin alphabet, without even needing to know what the sign means.



Figure 2.1: LIS sign for *Fun*, written in SignWriting. Extracted from (Di Renzo et al., 2012)

Section 2.1 provides an account of the history and the evolution of SignWriting over time, Section 2.2 focuses on the features of the system which are relevant for the present work. Section 2.3 describes an alternative SignWriting coding which has proven very useful during our work. The opportunities and the issues of digital SignWriting are covered in Section 2.4.

## 2.1 The history of SignWriting

Southern California, 1966. A young girl, who was in a professional ballet training class, invented an iconic writing system for recording ballet steps, for her own personal use. Her name was Valerie Sutton, and the system is presently known as *DanceWriting*. Using this system, each significant step of any dance can be recorded by writing a stick figure on a five-lined staff. Each line of the staff represents a specific body level, from the Foot Line (the bottom one), to the Shoulder Line (the top one). Fig. 2.2 shows a modern (1983) example of *DanceWriting*, extracted from (Sutton, 1983).



Figure 2.2: First steps of the *Sugar Plum Fairy*, from the second act of the *Nutcracker* ballet, recorded with DanceWriting (Sutton, 1983).

The first important “public” demonstration of DanceWriting dates back to 1970, as Sutton moved to Copenhagen. At that time, the world-renowned, historic ballet steps of the Royal Danish Ballet, called *The Bournonville Schools*, were at risk of being forgotten for lack of recording. Sutton used her personal writing system to record and preserve these historic dances, improving the system itself in the process (Sutton, 1996).

The success of Sutton’s efforts did not pass unnoticed. In 1974, SL researcher Lars von der Lieth, and his team at the University of Copenhagen, was looking for way to record signs and gestures. He asked Sutton to record the movements from a videotape, and she fulfilled her task, using a DanceWriting variation which basically recorded any movement from the waist up, with a few differences (Sutton, 1996). The foundations of *SignWriting* were being laid (Fig. 2.3).

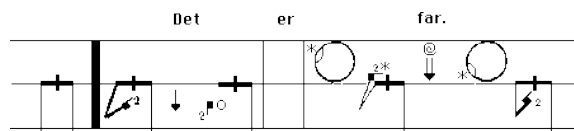


Figure 2.3: Early fragment of SignWriting (1976), extracted from (Sutton & von der Lieth, 1976). This fragment represents a Danish SL sentence meaning *It is father*. The significant influence of DanceWriting over the newborn SignWriting can be identified by the glyph shapes and by the presence of a three-lined staff.

During the second half of the 1970s, focus slowly shifted from DanceWriting to SignWriting, which underwent a gradual improvement process, fueled by the growing interest over the writing system.

In 1977, Judy Shepard-Kegl, a graduate student in linguistics at the Massachusetts Institute of Technology (MIT), arranged the first SignWriting Workshop in the United States with the New England Sign Language Society. The workshop was held at the MIT, and the ensuing debate brought up important issues. The discussion led in turn to a gradual reshaping of the system (covered later in this section).

In the same year, SignWriting was taught to a deaf community (the National Theater of the Deaf) for the first time. The success achieved with this experience consolidated the motivation of Sutton's team, bolstering the SignWriting dissemination activity.

*The early years were also filled with experiments. One day, Sutton received permission from a school in Manchester, New Hampshire, to visit one class of Deaf students for one hour. There, Sutton wrote the sign for hello on the blackboard in SignWriting. The students guessed it immediately. They became quite excited when they found they could read basic signs in a matter of minutes.*

(Sutton, 1996) Valerie Sutton - The History of SignWriting

The first paper about SignWriting (Sutton, 1977) dates back to 1977, and it was presented at the National Symposium on Sign Language Research and Teaching in Chicago. During her presentation, Sutton invited Dr. Stokoe to talk to the group as well.

Now, almost forty years later, SignWriting is one of the most used writing systems for SL, improved by the work of several research teams around the world, and by the daily usage of countless communities <sup>2</sup>.

As a consequence, years of work with SignWriting have resulted in a rich literary production, which embraces different fields and comes in different forms. Over time, most of the produced resources have been gathered in the *SignWriting Literature Project* which currently includes Wikipedia pages written in SL (Fig. 2.4), religious literature, SL works by deaf authors, SL dictionaries with definitions written in SL, and much more.

---

<sup>2</sup>The official list of the communities using SignWriting is available at <http://www.signwriting.org/about/who/>.

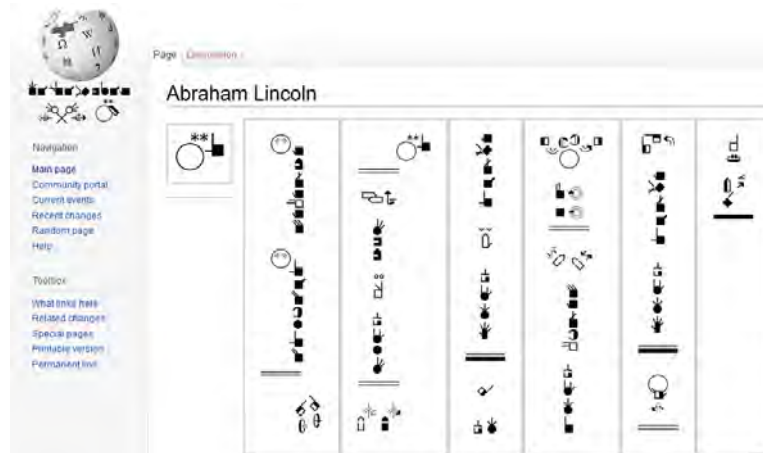


Figure 2.4: Detail of the ASL Wikipedia page about Abraham Lincoln, written in SignWriting.

SignWriting itself underwent deep changes. High-level aspects, such as the representation rules and the visual organization, evolved along with low-level aspects, such as the set of available glyphs and the appearance of the glyph themselves. Tab. 2.1 illustrates a number of major changes in SignWriting over time<sup>3</sup>.

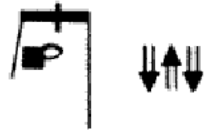



**Major changes to SignWriting since 1974**

Change Description	Sample
<p><b>1974 - Early Origins</b>                      SignWriting originally stemmed from DanceWriting in 1974. DanceWriting places a “stick figure drawing” on a five lined staff. Figures were written from left to right, facing the reader (receptive viewpoint).</p>	
<p><b>1974 to 1980 - Detailed SignWriting</b>                      The five-lined staff was replaced with a three-lined staff for the upper body. Tiny numbers were used to show which fingers were projecting. Each sign was separated by a thin vertical line. A thick vertical line began and ended each sentence.</p>	

*Continued on next page*

<sup>3</sup>The details of the evolution of SignWriting during the last forty years is far beyond the scope of this work, further information is available at <http://www.signwriting.org/library/history/hist008.html>

**Major changes to SignWriting since 1974** – *Continued from previous page*

Change Description	Sample
<p><b>1980s - Full-Body SignWriting</b> The three-lined staff was discarded due its limited usefulness. Tiny numbers for fingers discarded, each hand configuration was given a special appearance.</p>	
<p><b>1980s - SignWriting Handwriting</b> The stick figure was discarded: it was only used occasionally for special torso movements or contact with the hips or shoulders. The use of vertical lines to divide signs and sentences was limited.</p>	
<p><b>1986 to 1996 - “The Expressive Revolution”</b> Several deaf SignWriting staff members manifested the need to express their our own language from their own perspective. The composition rules were changed: the receptive viewpoint was replaced with the expressive viewpoint. In other words, users began writing signs as they are seen from the person who is signing, and not from the one who is watching.</p>	<p align="center">N/A</p>
<p><b>1986 to 1996 - “The Stacked Revolution”</b> More and more deaf people were choosing the SignWriting Handwriting, but instead of writing the symbols in a line from left to right, they were naturally stacking the symbols to look like the human body. Each sign was like a little “stacked unit”. The units were then placed from left to right on the page.</p>	
<p><b>1996 to present - “The Vertical Revolution”</b> Many deaf SignWriting staff members expressed the interest in writing down the page in columns. After the Stacked Revolution, since the visual organization of each sign was vertical, writing from left to right did not feel as natural (and consistent) as writing in columns.</p>	

*Table 2.1:* Major changes to SignWriting since 1974 to present. Information and images drawn from (Sutton, 1996).



As shown in Tab 2.1, SignWriting is constantly evolving over time. As a consequence, every few years (typically when a major change occurs), a new release of the SignWriting system is issued. Any digital artifact presented in this work has been produced using the 2010 version of SignWriting as a reference point. This was a common agreement of most SignWriting research teams in the computer science field around the world. The decision was taken in order to have a set of applications, produced by different teams, which can share each other's data. For further information refer to Section 2.4.



Figure 2.5: Communities which use SignWriting around the world (Sutton, 1996).

Over time, SignWriting reached a worldwide diffusion, both as a communication and as a SL teaching tool. As visible in Fig 2.5, each of the 5 continents is covered by a number of communities using SignWriting. The writing system is adopted by a large number of communities both in the Americas and in Europe, and it is also present (in minor extent) in Asia, Africa and Oceania. Tab. 2.2 provides an account of the communities adopting SignWriting within each continent.

#### Communities using SignWriting around the world

Americas	Europe	Asia	Africa	Oceania
Bolivia	Belgium	Japan	Ethiopia	Australia
Brazil	Czech Rep.	Malaysia	Malawi	New Zealand
Canada	Denmark	Nepal	Tunisia	
Colombia	Finland	Philippines	South Africa	
Mexico	France	Saudi Arabia		
Nicaragua	Germany	Taiwan		
Peru	Great Britain			
United States	Ireland			
	Italy			
	Malta			
	Netherlands			

*Continued on next page*

**Communities using SignWriting around the world – *Continued***  
*from previous page*

Americas	Europe	Asia	Africa	Oceania
	Malta Northern Ireland Norway Poland Portugal Spain Sweden Switzerland			

*Table 2.2: Communities using SignWriting around the world, divided by continent. Information extracted from (Sutton, 1996).*

As shown in Tab. 2.2, SignWriting has spread in many countries, some of them being also very far from the “home” state of the writing system, i.e. California. Such wide diffusion has been mainly fostered by the SignWriting Literature Project for Deaf Students (Sutton, 1996). The project has developed software, web sites, DVDs, children’s stories and instruction materials, aimed at supporting the teaching of reading and writing in both SLs and VLs, using SignWriting. These materials are donated to schools with deaf students. One of the first schools to experiment with this method was the Hodgin Elementary School in Albuquerque, New Mexico, in 1999. Many other schools in many other countries have subsequently joined the SignWriting Literature Project for Deaf Students (Sutton, 1996). The teaching activities within classrooms with deaf students are supported through the above-mentioned teaching materials, training for teachers and parents, and full technical support. In order to fully understand the reasons leading a teacher with deaf students to apply for the SignWriting Literature Project for Deaf Students, it is worth reporting the following lines, written by the teachers at the Kasterlinden School for the Deaf in September 13, 2004 (Sutton, 1996) to describe their experience with the classroom.

***Why do you want to learn SignWriting?***

*SignWriting is the written form of SignLanguage. We want to teach the children the difference between the two languages they live with, Flemish sign Language (FSL) and Dutch (written or spoken). If we want to teach the children FSL we would want to write it down, it would be a mistake to write it in Dutch but with the grammar of FSL. SignWriting is also very visual, for ex: if you want to actually show the children the difference between two sign. They can mark it on the board!*

***What have been some of your past frustrations when teaching?***

*Not being able to have something on paper about FSL. The children forgot what I told them about FSL because they had no means of remembering, like writing it down. The children didn't really think FSL was a full language because they were always given Dutch, in books and on paper.*

(Sutton, 1996)

Statistical data about the actual number of SignWriting users is not currently available, since no study has been attempted in such direction. The concept of *SignWriting user* itself is vague and needs some clarification. At least three categories of people can be classified as SignWriting users, according to their frequency of use and involvement with the writing system.

- People who actively use SignWriting.
- People who actively used SignWriting (most likely at school), but they no longer do.
- People who have seen SignWriting, they know “something” about it, but they cannot read it.

As an additional information, however, it is worth reporting that an informal conversation about this topic has been held between members of the SignWriting staff (including Valerie Sutton) and members of the SignWriting research community. Each participant had a clear picture of the diffusion of SignWriting in his/her own country, in terms of involved people. The following figures emerged: the first group (people who actively use SignWriting) can be esteemed in the order of tens or below (not reaching the count of ten in some countries, such as Nicaragua) per country. The second group (people who actively used SignWriting, but they no longer do) can be esteemed in the order of tens, or a few hundreds per country. The last group, finally, can be esteemed in the number of hundreds per country. No further (or more accurate) claim is possible about this topic.

## **2.2 Overview of SignWriting system**

As introduced in the opening of this Chapter, SignWriting is a featural writing system for SL, which relies on small iconic elements - the glyphs, to represent any sign.

SignWriting can be written both from the expressive viewpoint, i.e. the point of view of the person producing the signs, and from the receptive viewpoint, i.e. the point of view placed in front of (and facing) the person producing the signs (Fig. 2.6).

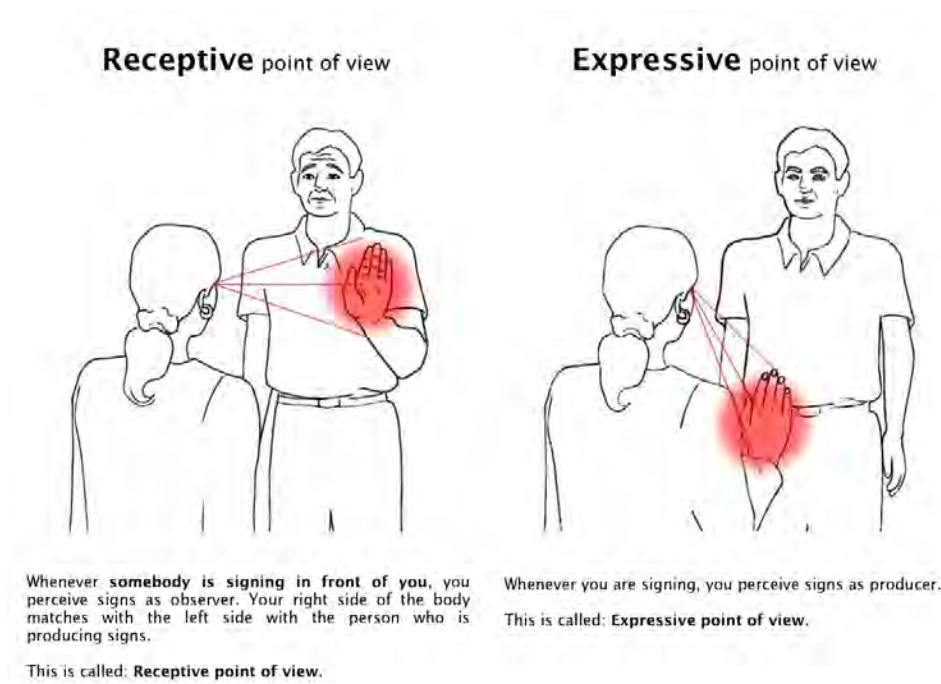


Figure 2.6: Difference between expressive and receptive point of view. Extracted and translated from (Di Renzo et al., 2012).

The modern official SignWriting standard, however, recommends the use of the expressive viewpoint, since the composition process turns out more natural for SignWriting users (Sutton, 1996). Adopting the expressive viewpoint, any glyph which is written on the left side of a sign describes the left side of the body of the producing person.

Since 1996, the SignWriting standard also recommends to write following a vertical organization: writing signs in columns, one below the other, as shown in Fig. 2.4. For this reason, most of the SignWriting texts shown in this work are written adopting a vertical organization.






The set of movements and positions that a human body can produce from the waist up is huge. As a consequence, the set of glyphs that SignWriting provides to write down any sign is accordingly vast (approximately 45.000 units). The whole set of glyphs is referred to as the International SignWriting Alphabet (ISWA). As introduced in Section 2.1, the 2010 version of the ISWA: International SignWriting Alphabet 2010 (ISWA-2010) is the most recent one, and it has been adopted as the current standard by many research teams around the world, including ours <sup>4</sup>.

The ISWA-2010 organizes its glyphs by dividing them into 7 *categories*. The categories are identified by following a very intuitive principle: each one covers a different anatomic part of the human body, with a small number of exceptions. Tab. 2.3 provides the full list of categories, illustrating each

<sup>4</sup>For the full ISWA-2010 reference, please refer to (Slevinski, S.E., Jr, 2010a)

entry with one of its most representative glyphs.

### ISWA-2010 Categories

Category Description	Glyph
<p><i>Category 1: Hands</i> Hand configurations from over 40 SLs, divided into 10 groups (numbered 1-10). The groups are identified by the fingers involved in the configuration. <b>Example:</b> any configuration featuring a single index finger extended is located under <i>Group 01: Index</i>.</p>	
<p><i>Category 2: Movements</i> Contact symbols, movements for fingers, hands, arm, forearm etc. divided into 10 groups (numbered 11-20). The groups are identified by the geometric plane to whom they are parallel. <b>Example:</b> any straight movement parallel to the Front Wall Plane (the vertical plane identified by the front wall) are located under <i>Group 16: Straight Wall Plane</i>.</p>	
<p><i>Category 3: Dynamics</i> Symbols which can be placed beside movements to specify their dynamics and coordination. They are all gathered in <i>Group 21: Dynamics &amp; Timing</i>.</p>	
<p><i>Category 4: Head &amp; Faces</i> Facial expressions, head and face movements, divided into 5 groups (numbered 22-26). The groups are identified by the anatomic part which produces the movement or the position, ordered from the top of the head to the bottom of the neck. <b>Example:</b> movements and positions produced with mouth and lips are located under <i>Group 25: Mouth Lips</i>.</p>	
<p><i>Category 5: Body</i> Symbols for movements and positions of torso, shoulders, hips, limbs etc. divided into 2 groups (numbered 27-28). <b>Example:</b> movements and positions produced with the trunk of the body are placed under <i>Group 27: Trunk</i>.</p>	

*Continued on next page*

ISWA-2010 Categories – Continued from previous page



Category Description	Glyph
<p><i>Category 6: Detailed Location</i> Used in ordered lists as annotation outside of the written cluster. Not used for everyday writing. They are all gathered under <i>Group 29: Detailed Location</i>.</p>	
<p><i>Category 7: Punctuation</i> Punctuation symbols, used when writing complete sentences or documents in SignWriting. They are all gathered under <i>Group 30: Punctuation</i>.</p>	

Table 2.3: Categories identified within the ISWA-2010. Each category is illustrated with one of its most representative glyphs. Information and images drawn from (Sutton, 1996) and (Slevinski, S.E., Jr, 2010a).

Figure 2.7 shows the LIS sign for *Fun* (already introduced in Fig. 2.1) highlighting each glyph with a color, depending on its category. From top to bottom, the categories are: Head & Faces (1 glyph), Hands (2 glyphs), Movements (2 glyphs) and Dynamics.

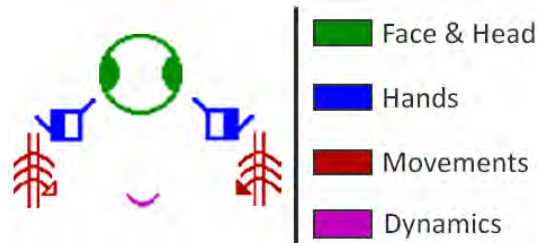






















Figure 2.7: LIS sign for *Fun*, written in SignWriting. Different colors highlight glyphs belonging to different ISWA categories.

of a further specification layer within the ISWA. In fact, each category is in turn divided in one or more *groups*. Groups help in keeping a logical and linguistic organization within categories which would otherwise be too vast to manage. Category 1: Hands alone counts over 20.000 units, and it will be our running example throughout this section to actually show how the ISWA is organized. Category 1: Hands is divided in ten groups, each identified by the fingers involved in the hand configuration. Tab 2.4 shows the full list of groups within Category 1: Hands, each entry is illustrated with one of its most representative glyphs, and the relative hand configuration.

### Organization of Category 1: Hands

Group Number	Group Name	Glyph	View
Group 01	Index		
Group 02	Index Middle		
Group 03	Index Middle Thumb		
Group 04	Four Fingers		
Group 05	Five Fingers		
Group 06	Baby Fingers		
Group 07	Ring Finger		
Group 08	Middle Finger		
Group 09	Index Thumb		
Group 10	Thumb		

*Table 2.4:* Group organization of Category 1: Hands, within ISWA-2010. Each group is illustrated with one of its most representative glyphs, and the relative hand configuration. Information and images drawn from (Sutton, 1996).

Similar distinctions are present in each category of the ISWA. The full list

of groups is available in Tab. 2.5.

### Glyph groups in ISWA-2010

Category	Group(s)
Category 1: Hands	Group 01: Index Group 02: Index Middle Group 03: Index Middle Thumb Group 04: Four Fingers Group 05: Five Fingers Group 06: Baby Finger Group 07: Ring Finger Group 08: Middle Finger Group 09: Index Thumb Group 10: Thumb
Category 2: Movements	Group 11: Contact Group 12: Finger Movement Group 13: Straight Wall Plane Group 14: Straight Diagonal Plane Group 15: Straight Floor Plane Group 16: Curves Parallel Wall Plane Group 17: Curves Hit Wall Plane Group 18: Curves Hit Floor Plane Group 19: Curves Parallel Floor Plane Group 20: Circles
Category 3: Dynamics	Group 21: Dynamics & Timing
Category 4: Head & Faces	Group 22: Head Group 23: Brow Eyes Eyegaze Group 24: Cheeks Ears Nose Breath Group 25: Mouth Lips Group 26: Tongue Teeth Chin Neck
Category 5: Body	Group 27: Trunk Group 28: Limbs
Category 6: Detailed Location	Group 29: Detailed Location
Category 7: Punctuation	Group 30: Punctuation

Table 2.5: Glyph groups in ISWA-2010.



We have shown that ISWA is organized in categories, and categories in turn are organized in groups. There is no further specification layer within the groups: each of them contain a set of glyphs. However, each group carries a set of rules (variations, covered later in this section) that must be respected by each glyph it contains. As an example, Group 01: Index under Category 1: Hands strictly contains hand configurations featuring a single extended index finger, and such glyphs follow the same rules.

Within most groups, we can identify a number of subsets of glyphs which are “similar”. Each subset can be summarized by its most representative glyph, namely the *base symbol*, which acts as a prototype (see Fig. 2.8) for the whole subset. Applying simple graphical transformations (e.g. rotations, changes in the color fill, etc.) to a base symbol, we can obtain any other element in the subset of the base symbol. Such graphical transformations, namely *variations* (according to SignWriting terminology), when applied to a given glyph, reflect a slight change (rotation, orientation, etc.) in the production of the matching body movement or position. The available variations vary from group to group.

Resuming the above example of Category 1: Hands, we use a base symbol (Fig. 2.8) from Group 1: Index and show its possible variations. Any glyph within the same group (or in the same category, in some cases) has the same possible variations, in our case, the variations available to the glyph in Fig. 2.8 are shared among the whole category (Category 1 - Hands).

Three variations are available for the glyph in Fig. 2.8. According to which

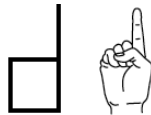


Figure 2.8: Base symbol of a hand configuration featuring a single index finger extended. Extracted from (Sutton, 1996)

part of the hand is visible to the signer, there are three possible *orientation* variations (Fig. 2.9): palm (white), cut (black and white), back (black). According to the geometric plane which is parallel to the configuration, two possible *plane* variations (Fig. 2.10) are available: horizontal (i.e. the hand is parallel to the pavement), vertical (i.e. the hand is parallel to the front wall). Finally, any glyph of the hand has 8 possible variations according to its *rotation* (Fig. 2.11):  $0^\circ$  (upright hand),  $45^\circ$ ,  $90^\circ$ ,  $135^\circ$ ,  $180^\circ$ ,  $225^\circ$ ,  $270^\circ$ ,  $315^\circ$ .

A number of different variations is available within the other categories and

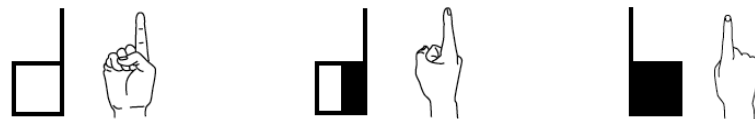


Figure 2.9: Orientation variations available for a glyph belonging to Category 1 - Hands. Extracted from (Sutton, 1996)

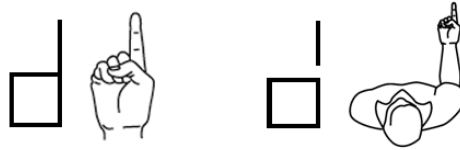


Figure 2.10: Plane variations available for a glyph belonging to Category 1 - Hands. Extracted from (Sutton, 1996)

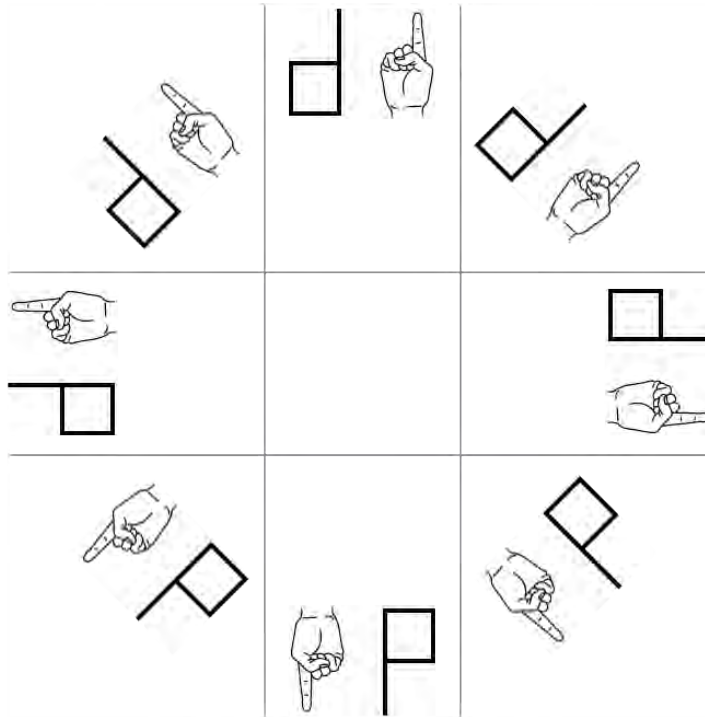
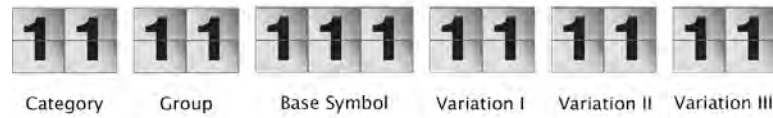


Figure 2.11: Rotation variations available for a glyph belonging to Category 1 - Hands. Original configuration image extracted from (Sutton, 1996)

groups, but the underlying logic remains the same. For further information please refer to Sutton (1995) and Sutton (1996).

Categories, groups, base symbols and variations are the key elements to identify a unique code (ISWA code) for each of the 45.000 glyph within the ISWA. Such identifier is formed by 13 digits and it is built by encoding the linguistic and production features of a glyph. As a consequence, it is able to identify the glyph in a univocal way. Fig. 2.12 shows the details of the ISWA code, and the involved elements for each group of digits. The first pair of digits identifies the top level set, i.e. the category. The second pair identifies the group. Digits 5 to 7 identify a specific base symbol. The last three pairs of digits identify the three variation available for the glyph.

The ISWA code is a key element for the digitalization of SignWriting, since it is much easier for a machine to work with 13-digit codes, rather than with raw unorganized symbols. See section 2.4 for more information about digital SignWriting.



*Figure 2.12:* Illustration of the 13-digit Sutton ISWA code used to uniquely identify any SignWriting glyph.

## 2.3 A different SignWriting coding

SignWriting has been adopted as a writing system for LIS by the Sign Language and Deaf Studies (SLDS) laboratory at the Institute of Cognitive Sciences and Technologies (ISTC) - National Research Council (CNR) in Rome. Between 2007 and 2012, during the preparation of her Ph.D. thesis (Bianchini, 2012), Bianchini analyzed the way deaf and hearing members of the SLDS were using and mastering SignWriting.

It is worth mentioning that the members of SLDS learned SignWriting on their own, by studying on the 1995 SignWriting manual (Sutton, 1995) and using a digital editor provided by Sutton (Sutton, 1996). Despite a very good knowledge of SW, recurring problems were noticed in its use. Most problems were due to the lack of strict consistence in the organization of SignWriting that was evident both in the manual and in the digital editor in use in the lab. As an example, by analyzing the glyphs for the movements of the hands, the following anomalies were detected:

- Variations of the same base glyph do not describe the same trajectory.
- It is not possible to realize the same trajectories on every geometric plane (Fig. 2.13).

Therefore, Bianchini carried out a complete reorganization of SignWriting, in full compliance with the work of Sutton and her team (i.e. no glyph has been deleted). As a matter of fact, Bianchini suggested additions (see Fig. 2.13) in order to increase the consistence of the system (Bianchini & Borgia, 2012). This reclassification required a new numbering system for the glyphs, and, as a consequence, a different ISWA coding, hence named International SignWriting Alphabet Bianchini (ISWA-BIANCHINI). Such coding holds different advantages, which are discussed in depth in (Bianchini, 2012). The most relevant advantage, however, for both linguists and computer scientists working with SignWriting, is that the ISWA-BIANCHINI coding allows to easily extract the different features of the glyphs, with no exception. This makes querying the tens of thousands of glyphs much easier. For instance, a single query may extract all, and only, the movement from right to left of the right hand in the horizontal plane.

For the above reason, the business logic of any digital artifact shown in the present work adopts the ISWA-BIANCHINI coding for SignWriting glyphs, when querying and matching becomes necessary.

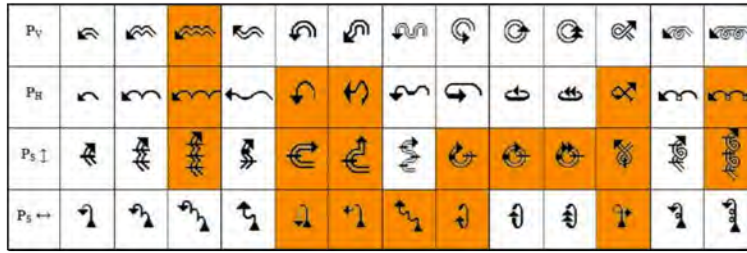


Figure 2.13: An example of the improvements introduced by ISWA-BIANCHINI. The white boxes show the possible trajectories of hand movements within ISWA2008, arranged by geometric planes; The orange boxes show the missing trajectories which were added to improve the system consistence.

## 2.4 Digital SignWriting

A pencil and a piece of paper are the only required items to produce signs using SignWriting. In fact, ASL signs were written by hand with ink pens from 1974 to 1986 (Sutton, 1996). Since the early years, anyway, the need to produce a digital version of the system was evident. The informatisation of the system, started in 1986 by Richard Gleaves with the SignWriter computer program (Sutton, 1993), allowed SignWriting to achieve a wider diffusion through the press of newspaper, books, and, in the last decade, websites and other digital resources. The glyphs, which before 1986 were exclusively handwritten, were joined by their digital duplicates, and the organization was ensured by assigning to each of them a unique ISWA code.

### 2.4.1 Digital resources in SignWriting

SignWriting has been employed to produce different digital resources, over time. The present section is not intended to be a complete list of all the digital artifacts featuring SignWriting, it is rather intended to provide an idea of the diversity of such artifacts. All the digital resources shown in this section are bilingual, they support English and ASL, thus they ensure full accessibility both to hearing and deaf people.

#### 2.4.1.1 Websites

The SignWriting Website (Fig. 2.14) is the reference site for SignWriting users and researchers, maintained by Valerie Sutton since 1996. It is an example of website which is accessible to deaf and hearing people alike. The website navigation elements carry labels both in English and in ASL, and the content is always available in both languages too. It also provides a number of utilities for SignWriting users, such as an online digital editor to produce signs (see Section 2.4.1.4).



Figure 2.14: Home screen of the *SignWriting Website* (available at <http://www.signwriting.org/>).

### 2.4.1.2 Blogs

Adam Frost, a deaf interpreter from the U.S.A. built and currently maintains his blog *The Frost Village*. The blog features two languages: English (through alphanumeric characters) and ASL (through SignWriting). Mr. Frost himself explains the rationale of his work on the blog, using the following words: “Well, as I have said, I am Deaf, so I use ASL. I strongly believe that there should be a way to write what I sign, just like anyone can write what they speak. SignWriting does just that. My webpage is proof of that technology can support it to even be in printed on the web.”. The two supported languages are not available together at the same time (like The SignWriting Website), but the user has the faculty to switch from one to another anytime.



Figure 2.15: Home screen of the *Frost Village* blog (available at <http://www.frostvillage.com/>).

### 2.4.1.3 Wikipedia

The ultimate goal of the ASL Wikipedia Project is to provide a bilingual, educational, informational tool. The ASL Wikipedia is intended to “provide information about the world to deaf and hearing people who use ASL as their daily, primary language”. The produced artifact is actually a tool for teaching literacy in two languages: American Sign Language and English. The ASL articles in the online ASL Wikipedia are translations of the English articles in the English Wikipedia. The vision of the developers foresees a bilingual learning as easy as “clicking on a tab in your web browser to read the same article in two different languages” (Sutton, n.d.).

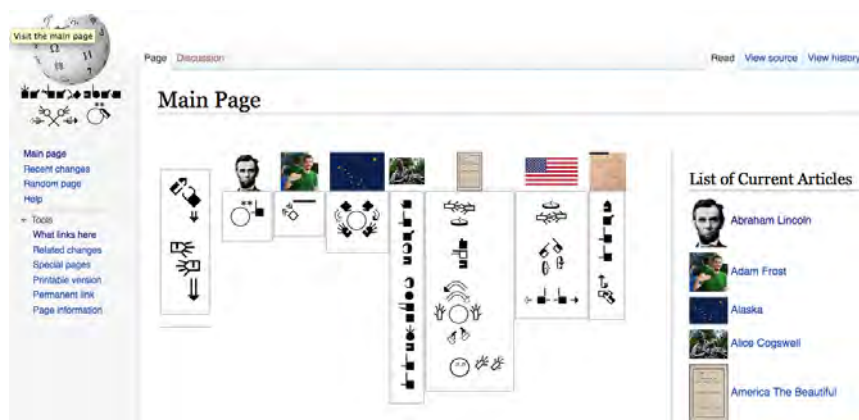


Figure 2.16: Home screen of the ASL Wikipedia Project (available at [http://ase.wikipedia.wmflabs.org/wiki/Main\\_Page](http://ase.wikipedia.wmflabs.org/wiki/Main_Page)). A subset of the available entries are visible.

### 2.4.1.4 Digital editors

SignWriting digital editors belong to a very important class of software. In fact, they are the tools that enable the creation of digital SL resources written in SignWriting. In other words, they are critical for the informatization of SignWriting. The first digital editor for SW, named “SignWriter” was developed in 1986 by Richard Gleaves (Sutton, 1993). Since then, many applications have been produced by different research teams, delivered in different ways, ranging from desktop to web applications (Borgia et al., 2014). A full list of such software is available at <http://www.signwriting.org/downloads>.

Most SignWriting digital editors basically provide the same functionalities. Despite differences in design and implementation existing from one editor to another, such functionalities are:

- Search for (or type) glyphs which belong to the ISWA.
- Insert the chosen glyphs onto an area which is designated for the composition of the sign.

- Manage the glyphs on the sign composition area.
- Save the sign in one (or more) formats.

Fig. 2.17 shows a screenshot of a SignWriting digital editor, the composition of the LIS sign for *Fun* shown in Fig. 2.1 is underway in the top-right part of the screen. More information about the editor displayed in Fig. 2.17 is available in Chapter 3.2.



Figure 2.17: Screenshot of *SWift* SignWriting digital editor (available at <http://visel.di.uniroma1.it/SWift/>).

## 2.4.2 Challenges of digital SignWriting

The informatisation of SignWriting poses different challenges. First of all, computer scientists need to devise effective and efficient ways of dealing with a set of symbols as large as the set of movements and positions that can be produced from the waist up. When designing a SignWriting digital editor, for instance, the large cardinality of the ISWA set might become a major problem for the application if addressed incorrectly, since it might affect both logic and presentation layers. It is necessary to get as close as possible to the *aurea mediocritas* between the unrestricted access to the data (the glyphs) and the presentation of a human-manageable amount of information. As an example, if a user is looking for a particular glyph, belonging to Category 01 - Hands, it is unreasonable to present him/her the whole lot of glyphs within the category, since it contains more than 20.000 symbols.

Another important fact that computer scientists must take into account when working with SignWriting is the nature of the ISWA code. As mentioned in Section 2.2, the ISWA code is constructed by taking exclusively into account linguistic and production features of the glyphs. This means that glyphs having a very similar ISWA code may have a very different appearance. Such detail might require special attention when producing SignWriting applications based on image-processing techniques (see Chapter 5). Last but not least, SignWriting grants a very high degree of freedom to

its users. The rules which Sutton sets in (Sutton, 1995) merely cover all the available categories and groups, specifying how to write glyphs, their meaning, and their possible variations. There is no high-level rule about the composition itself, no restriction is set to the number of glyphs within a sign, to their possible spatial arrangement and to their relative positioning. It not difficult, for example, to find signs written with two heads (used sometimes to specify movement), or signs with no head at all (used when the movements or expressions of the head are not important for the representation of the sign). Given this, it is very difficult to produce a strict modeling of SignWriting compositions, as a consequence, there is currently no way of validating such compositions.

The features of SignWriting, and the challenges they pose to the informatisation of the system, have been taken into account during the design of our applications. Such features directly influenced the choice of the methodologies and techniques we adopted to achieve the informatisation of the system, as reported in the following chapters.



## Chapter 3

# SWord

### 3.1 The SWord framework

Our efforts for the informatisation of the written form of SLs using SignWriting are carried out within the framework that we named SignWriting-oriented resources for the deaf (SWord). The overall goal of SWord is to make SignWriting effectively exploitable as a communication mean and as a learning support for deaf people. As computer scientists, we focused on making SignWriting viable as a writing system for SL in the digital world. As already explained in Section 1.2, even among HCI experts, it is often possible to find a “thick coat” of misunderstanding about accessibility design for deaf people. The “sharp” sound of the *SWord* acronym represents, in a metaphorical way, our efforts along the past four years, to clarify such misunderstandings and to overcome the resulting limitations (see Section 1.2 for further information).

Since SWord directly concerns SLs, we often found ourselves dealing with issues related to linguistic aspects of SLs and technical questions about SignWriting, which are far beyond our field of expertise. For this reason, the studies and the digital artifacts presented in this work have been designed with the aid of a multi-disciplinary range of competences. Such competences include: linguists, interpreters, SignWriting experts and psychologists and were provided by the research team at ISTC-CNR, and by Claudia S. Bianchini.

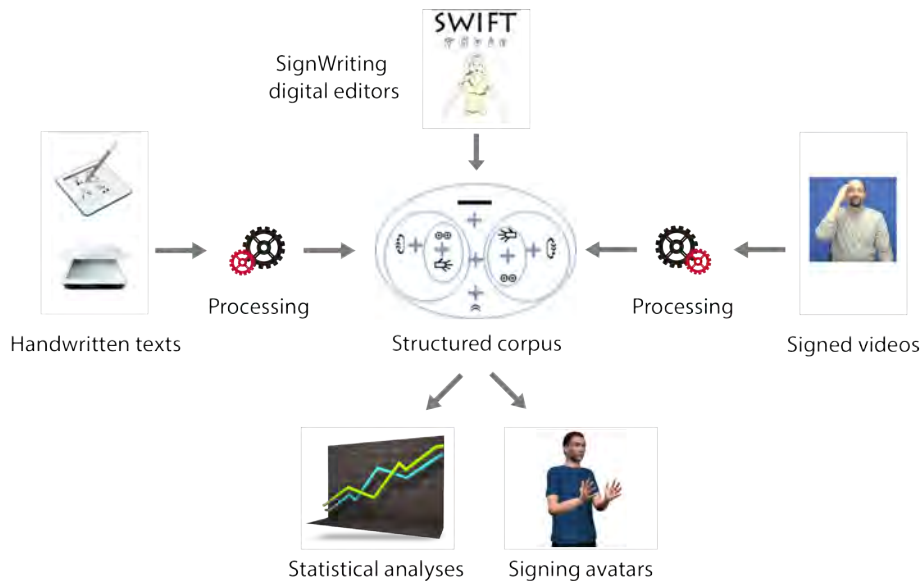
An intermediate goal of the framework is to gather, from different sources and through different acquisition methods, a significant amount of signs, to be stored in electronic form together with their decomposition into glyphs. Such set of signs is referred to as *structured corpus*. The purpose of a corpus prepared in this way is to allow the identification of recurring patterns in the composition of the signs, and the computation of relevant statistics on the transcribed form of the signs. Working with linguists in the past four years, we are well aware of the key importance of such statistical data, since they can help to gain understanding of the rules of SLs.

Fig. 3.1 shows a concept of SWord by Prof. Maria De Marsico. We designed

the framework to acquire its corpus of signs from 3 possible sources (Fig. 3.1, top and center part). First of all, digital editors: they are the tools that have been serving the purpose of producing digital signs written in SignWriting for about 30 years. In order to enable the acquisition of signs from a digital editor (and for a number of other reasons), we implemented our own SignWriting digital editor, namely SWift (illustrated later in this chapter). The advantage of using SWift is that it is already able to produce signs coded in a form which is best suited for the analysis, without the need for further action. As a consequence it can be used for a first phase of mass production of the structured corpus.

From this starting point, our aim is to implement a software environment that is also able to handle signs coming from other sources, such as handwritten documents in SignWriting or video clips of signing people. However, these sources require a substantial pre-processing. In the case of a handwritten document, it must be scanned and stored in an electronic form. In addition, each text must be processed to extract the signs and glyphs that compose it. To this end we designed a recognition engine for handwritten SignWriting symbols, which is the core topic of this work and it is covered in Chapter 4, 5, 6 and 7.

The last acquisition method requires the processing of video sequences of



*Figure 3.1:* Concept illustration of the SWord framework (courtesy of Prof. Maria De Marsico).

signing subjects, in order to identify the signs, and to represent them using SignWriting. Even in this case, the ultimate goal is to produce the same type of information (i.e. stored with the output format of SWift) gathering data from signed videos regarding both the performed signs and their composition. The pattern recognition procedures involved in this latter case are much more complex, and therefore it is not sure that we can achieve the

complete goal of allowing fully automatic gesture recognition. Nevertheless, we are planning to implement a “user assisted” procedure to simplify the system task, if needed.

As mentioned above, the structured corpus can be very useful for linguistic research and statistical analyses, nonetheless, as computer scientists, we are planning to use it also for a different goal, which is equally valuable. The idea is that the precise production information stored by each SignWriting glyphs allows to use them to determine the movements and expression of a signing avatar in a very accurate and satisfactory way. A similar approach has been already adopted by Karpov and Ronzhin (2014), by implementing a *3D signing avatar* for Russian sign language, which is based on signs represented with HamNoSys (introduced in Section 1.1.2). In this way, one might avoid using written text, which is almost impossible to automatically translate in SL, to derive the avatar behavior. Using the intermediate form of SignWriting as an alternative starting point to guide the avatar, people using Sign Language may be supported in a number of activities that would otherwise require the use of a VL (e.g. e-learning), even without directly knowing SignWriting. Moreover, the use of avatars in signed digital communication is often considered a solution to privacy issues that may be raised by the participation of a human subject. Their limited diffusion is due both to the difficulties related to their implementation, but also to the fact that avatar systems presently available do not achieve the expressiveness and naturalness desired by deaf people, which often have a very critical opinion about them (Kipp, Nguyen, Heloir, & Matthes, 2011). Our approach would address both problems. The description of the SWord framework is continued by introducing SWift, a digital editor for SignWriting.

## 3.2 SWift

The present section contains an overview of SignWriting improved fast transcriber (SWift) (Borgia, 2010), a digital editor which allows to compose written expressions in any SL in the world using SignWriting. It is important to remark that SWift has been produced as a result of a previous work by the author (Borgia, 2010). As a consequence, its design and development are not part of the present work, but the usability testing of the application, and a number of later added features are. Moreover, knowing the basics about SWift is mandatory to correctly understand the content of the following chapters in the present work, and to get a full picture about our efforts towards the digitalization of SignWriting.

### 3.2.1 Introduction to SWift

In order to develop a better tool (with respect to its competitors), we started by analyzing the existent SignWriting digital editors to detect any issues within their interface and their interaction modality. More specifically, we focused on the official SignWriting digital editor produced by the Sutton

team, i.e. SignMaker (SM) (Slevinski, S.E., Jr, 2010b), shown in Fig. 3.2. Our evaluation of SM was carried out by performing an expert cognitive walkthrough (Wharton, Rieman, Lewis, & Polson, 1994) of the interface and by collecting some informal opinions of people that have been using SM for a long time. During our analysis, we detected a number of issues that possibly make the user interaction with the application rather cumbersome, especially for novice users. The results of our analysis are discussed in full depth in (Bianchini, Borgia, Bottoni, & De Marsico, 2012; Bianchini, Borgia, & De Marsico, 2012; Borgia, 2010), and they are summarized in the following.

First of all, a number of important issues affects the components devoted to the interaction between the user and the application, such as buttons and toolbars. We observed that the buttons contain expert-oriented iconic content, i.e. they can all be better understood by signing users with good SignWriting proficiency. The buttons also contain text labels, which are only presented in English language. This forces a deaf user to read text (which may be uncomfortable in itself, as explained in Chapter 1) in a possibly foreign language. Moreover, the glyph search panel does not provide any help to direct a SignWriting beginner to the glyph he/she is looking for. Glyphs belonging to different categories are placed aside with no distinction, and no graphical cue to symbolize the sign component they represent. The iconicity of the symbols often makes up for this lack, but in other cases it may be difficult to tell which component a glyph is representing. Finally, managing glyph simultaneously is not allowed: the user can interact (insert, edit, delete) with only one glyph at a time. This can make the sign composition process long, and ultimately frustrating.

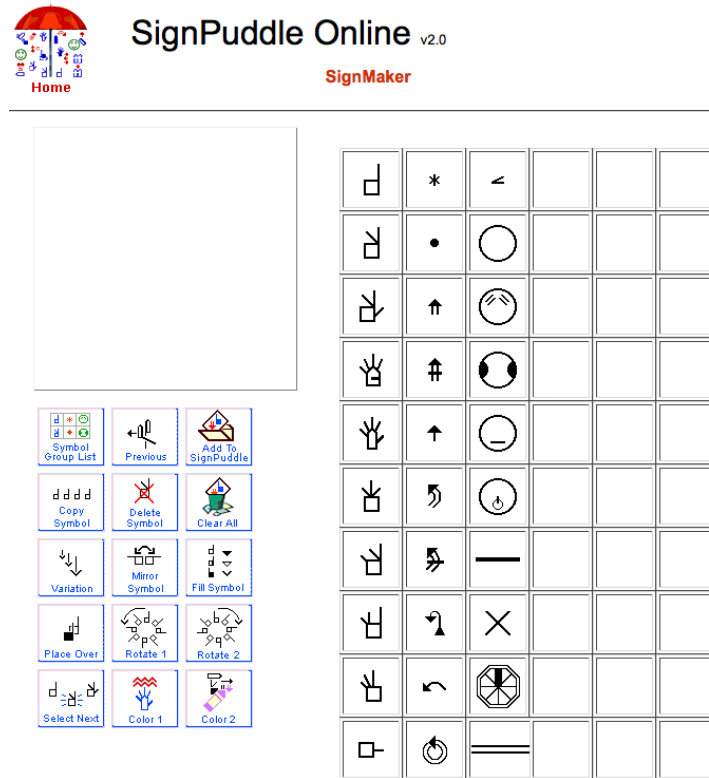


Figure 3.2: The user interface of Sutton’s SignMaker (Sutton, 1996).

We decided to use the results of our evaluation as a starting point for the production of SWift. Any design phase, from the early definition of the purpose of SWift to the latest graphical improvements, has been, and presently is, supported by a very active validation group, i.e. the research team at the ISTC-CNR in Rome, which includes many deaf researchers. Any choice is discussed and later checked with people who are a true sample of the group of main target users of SWift, in compliance with the core principles of User-Centered Design (Norman & Draper, 1986) and Contextual Design (Wixon, Holtzblatt, & Knox, 1990). The former design technique requires to give extensive attention to needs, preferences and limitations of the end-users of a product at each stage of the design process. The latter requires that researchers aggregate data from customers in the context where they will use the product, to apply these findings into the design stages.

The design and the implementation of the interface were performed according to a number of core principles, derived from both the specific purpose of the application and the general Human–Computer Interaction (HCI) guidelines, in particular:

- Intuitiveness: it is harmful to charge the user with the burden of learning the interface, when he can rather understand it. With this purpose in mind, each function is presented and used in a intuitive and familiar way.

- Minimization of information: each screen presents a small amount of essential information, to avoid overwhelming the user with a cluttered interface.
- Proper information format: icons are simple, large and familiar. If their meaning remains unclear, simple mouse-over-triggered animations have been embedded in the buttons definition to help the user. Dealing with text labels in the interface might be difficult for deaf people, so we kept these elements at a minimum (Perfetti & Sandak, 2000) and, whereas they proved necessary, they were accurately designed.
- User-driven interface testing: each change in the features and interface of SWift has been discussed with the team at the ISTC-CNR. The discussion involved high-level aspects (such as the logic behind the glyph search system), as well as low-level ones (such as the spatial placement of buttons within the interface).

### 3.2.2 Core features

SWift has been conceived to be a multi-platform application, and to have a wide and easy diffusion. The software has been developed as a self-containing web application, its interface can be easily included within a Document Object Model (DOM) element, thus allowing SWift to be embedded within any web-based platform, in order to provide a prompt SL support (Borgia et al., 2014). This also relieves the user from the burden of installation and configuration details. The first context in which SWift has been successfully integrated (Bottoni et al., 2013) is a Deaf-oriented E-Learning Environment (DELE). DELE is a visual E-Learning environment based on a metaphorical iconic representation of information (Bottoni, Capuano, De Marsico, & Labella, 2012).

Many functions of the application, such as the one enabling the search of a glyph within the ISWA, rely on data which are stored within the database of the application. Such data include:

- Glyphs: the single elements that are used to compose the signs. The information available on every single glyph record includes: the identifiers of the glyph (ISWA-2008, ISWA-2010 and ISWA-BIANCHINI), the path of its PNG image, its frequency of use (incremented every time that the glyph is used).
- Signs: whenever a completed sign is saved into the database, SWift records the association between the involved glyphs, their spatial coordinates and the title of the sign entered by the user. Such data are used to compute statistics which should provide the user with hints about co-occurring glyphs during the composition process, this innovative function will be detailed in Section 3.2.2.1.

### 3.2.2.1 Interface design

The features of SWift rely on a number of ideas which have been developed and implemented to make the sign composition process faster. Such ideas are illustrated analyzing the application by dividing its interface into 4 functional areas (Fig. 3.3).



Figure 3.3: SWift's (v. 1.01) home screen, divided in 4 functional areas.

Like any other digital editor, SWift provides an area to compose the sign. Such component is referred to as the *sign display*, and it is a whiteboard-resembling area whose purpose is to show the sign that the user is currently composing. A sign is composed by dragging a number of glyphs from the glyph menu and dropping them on the sign display. Once they are placed there, they become both draggable (to be relocated at will) and selectable (for editing purposes). Unlike SM, the whiteboard allows the selection of multiple glyphs at once, so that moving and editing more than one glyph at a time is possible, in order to save time and to avoid a frustrating experience to the user.

The *glyph menu* allows the user to search any glyph within the ISWA. Once the user finds the desired glyph, he/she can drag it and drop it on the sign display, in order to include it within the sign that he/she is composing. Most efforts were devoted to make the interaction with the glyph menu fast and effective. The underlying concept is basically “Why browse, when you can search?”, therefore the glyph menu features a search system which implements a completely different approach with respect to SM (discussed in Section 3.2.2.2). To support SignWriting beginners during the composition, the glyph menu has been designed to present a stylized human figure (Fig.

3.3, middle), named *puppet*, as a starting point for the search of a glyph. The *toolbox* contains functions for the management of the glyphs that are currently selected on the whiteboard. Such functions are devoted to glyph deletion, duplication, rotation and handwriting, the latter will be covered in Section 3.2.2.3.

The *hint panel* is one of the innovations that distinguishes SWift from other digital editors. This component shows in real-time, as the composition process is underway, a set of glyphs which are compatible with those the user already entered in the sign display. Compatibility is computed and updated according to the rate of co-occurrence of the glyphs. The glyphs suggested in the hint panel are immediately available to be inserted into the sign display. With such action, the user can save the effort and the time required to search for each glyph from scratch.

The compatibility check of the hint panel is triggered whenever a user changes the set of glyphs in the whiteboard (by inserting, editing or deleting glyphs). At that moment, the set of signs including such glyphs is retrieved from the database. The most recurring glyphs within the set of signs are presented in the hint panel in descending order of co-occurrence, taking into account a number of other minor factors.

The database underlying SWift is vital for the compatibility computation. For this reason, whenever a sign is composed and saved using any of the save procedures provided by the application, it is stored in the database too.

Actually, the hint panel offers features that are very similar to the predictive text input method for VLs, adopted by mobile phones and other electronic devices. Many studies, such as Curran, Woods, and O’Riordan (2006) demonstrate that predictive text is an important aid to communication when handling the set of characters of a VL, which are in the order of magnitude of tens. It is easy to realize how this can improve the interaction with a set of tens of thousands symbols.

### 3.2.2.2 Glyph search system

Reducing the composition time of a sign is the key to make SignWriting editors a viable alternative to the paper-pencil approach. Therefore, most efforts were devoted to the design and optimization of the glyph menu and of its navigation. SWift is intended for any SignWriting user, ranging from novice to proficient, so the *puppet* (Fig. 3.3, middle), has been designed to present in the most natural way the first important choice. As a matter of fact, each different anatomical part of the *puppet* (head, shoulders, breast, arms, hands) leads the user to a search interface for the specific glyphs related to that part. Four glyph families (contacts, punctuation, dynamics and coordination) are not available on the *puppet* because they do not belong to any specific anatomic area. They can be reached through 4 buttons located under the *puppet*.

In an ordinary SWift session, if a user is looking for a glyph belonging to a certain area, clicking on the corresponding anatomic part on the *puppet*



will lead to the related search screen. In particular, Fig. 3.4 shows the search interface related to the anatomic area of the hands, which is displayed after the user clicks on the hands of the puppet. Such window will be our running example in the present section to show the glyph search system in action.

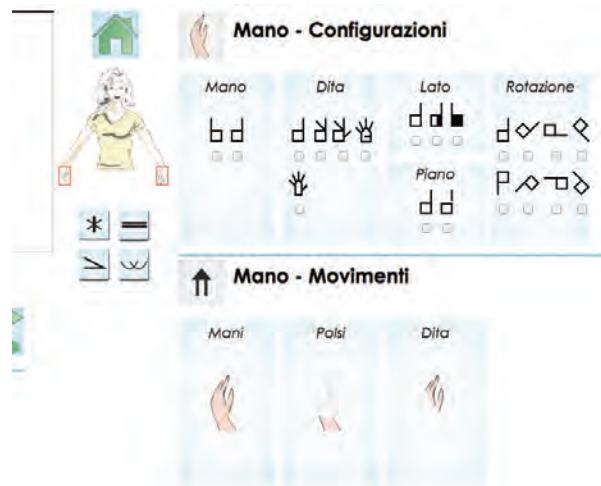


Figure 3.4: Search interface related to the anatomic area of the hands.

In this, and in any other search window, a narrowed version of the puppet (and of the buttons beneath) persists on the upper-left part of the screen, to form a consistent navigation menu which allows the user to switch from one anatomical part to another, without returning to the home. The current part (in this case hands) is highlighted on the puppet to support user orientation.

The blue boxes in the center of the screen are referred to as *choose boxes* (or simply *boxes*), they allow the user to search for the desired glyph. The boxes are organized in groups. Each group corresponds to a specific ISWA category and it is displayed under an icon and a text label explaining which kind of glyph can be found using the group.

Each box allows the user an exclusive choice corresponding to the selection of a core feature of the searched glyph. A choice can be performed for each box in a group. As explicitly asked by the team of ISTC-CNR, since users may have different mental organization while searching, they should be free to select the features of their glyph in any order, so the interaction with the choose boxes has been designed accordingly. A user might want to select the rotation of his glyph first, while another could select the number of fingers involved, or the hand (left, right) involved.

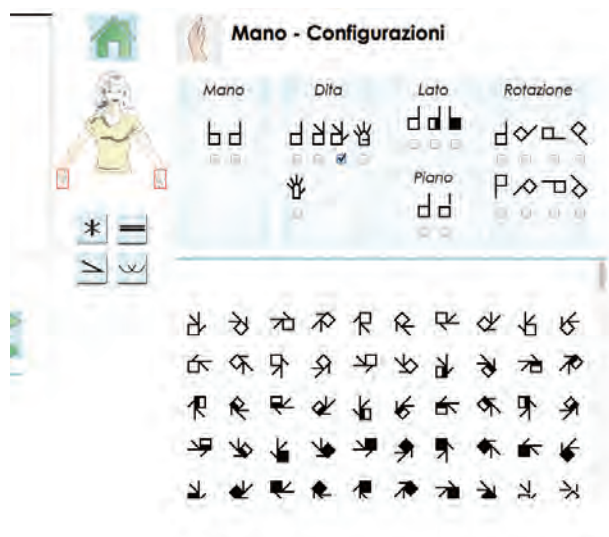


Figure 3.5: Search interface related to the anatomic area of the hands: one criterion (glyphs involving only three extended fingers) is selected in one of the choose boxes.

Once a choice is made in any of the boxes belonging to a certain group, the interface shows a set of glyphs meeting the requirements expressed so far. As an example, in (Fig. 3.5) the interface shows all glyphs containing three fingers, as requested by the selection in the second box. They are placed in a scrollable panel and they can be immediately dragged on the sign display. The panel replaces the group of boxes which cannot be select anymore (they correspond to other categories). In some lucky cases, the user is able to find the desired glyph just selecting one feature. More often, this happens after about 3 choices. After each new choice, the set of proposed glyphs is updated (narrowed). The user is granted the faculty of undoing his choices. Even in this case, once undone, the set of proposed glyphs is updated accordingly (enlarged). The choose boxes and the search criteria are designed so that, in the worst case (i.e. the user has given a value to each box) the user will search his glyph in a set of 40 glyphs maximum, a task that can be accomplished by a human in an acceptable time.

### 3.2.2.3 Support for user-defined glyphs

Although the ISWA contains tens of thousands of symbols, it is not rare to see a deaf person inventing a new *ad hoc* glyph (for an in-depth analysis of this phenomenon see (Bianchini, Borgia, & Castelli, 2011)). This practice is more often observed when a person achieves a good mastery of SignWriting. This allows him/her to draw glyphs which are not present in the ISWA, still respecting the rules of SignWriting. Such compliance is demonstrated by the fact that the appearance of the *ad hoc* glyphs is consistent with the rest of the ISWA, and that they are easily understandable by other SignWriting users (Bianchini et al., 2011).

With paper and pencil, inventing *ad hoc* glyphs is not a problem, since they

are drawn just like any other one. The challenge arises when dealing with a digital writing. As a matter of fact, SM does not support this practice in any way, so during SM sessions, the *ad hoc* glyphs are created by artificially juxtaposing existing glyphs of different shapes, and often belonging to totally different and discordant areas, in order to just create the desired visual effect. This has negative consequences over statistical and linguistic research (e.g. analysis of the frequency of the glyphs).

SWift specifically supports the creation of *ad hoc* glyphs by including a function to address this need. The handwriting interface (Fig. 3.6, left) is accessible from the toolbox and shows the whiteboard enlarged by a 1.5 factor to allow the user to draw with more accuracy.

The enlarged whiteboard includes, in light gray, the glyphs that the user

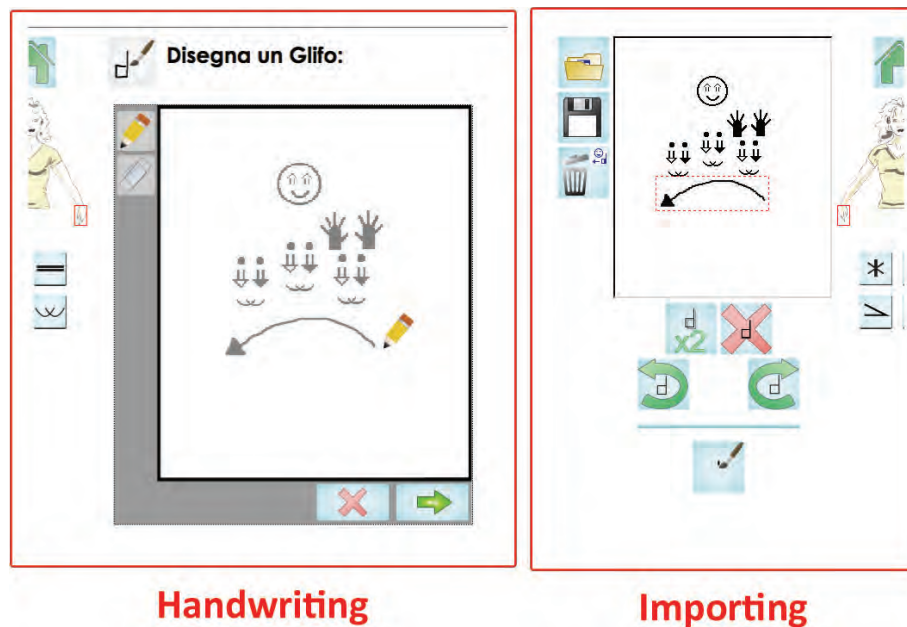


Figure 3.6: Support for handwritten glyphs provided by SWift: the glyph is first handwritten (left) and then imported on the sign display (right).

already placed on it, to help avoiding to draw an under-sized or over-sized glyph. At the end of the composition process, the glyph appears on the whiteboard (Fig. 3.6, right) together with the others. The *ad hoc* glyphs are saved in a special glyph family within the database. They are intended to be reviewed by research personnel and developers in order to support a constant update and widening of the ISWA.

#### 3.2.2.4 Toolbox design

Buttons deserved particular attention. We avoided including text labels, while devising a strategy to help the user understanding their function. To this aim, each one is connected to an animated sequence of a few seconds that activates on mouse-over. Each sequence animates the icon of the cor-

responding button, making the associated function more understandable. The choice to use an animation, rather than a short clip with the signed sequence, is due to the high cost, in terms of space and loading time, of a signed sequence. Additionally, the functions associated with the buttons are fairly easy to “illustrate” using only icons and animations. Think, for example, about the functions for saving, rotating, and deleting a glyph. To get an idea of the result, see Fig. 3.7. It contains screenshots of animation frames of two buttons within the interface of SWift, namely *counter-clockwise rotation* and *delete*.

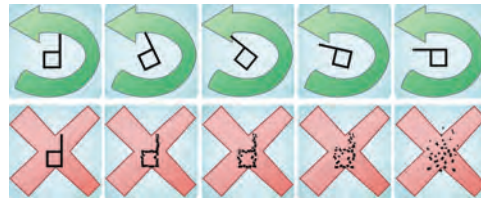


Figure 3.7: Animation frames of two buttons within the interface of SWift: *Anti-clockwise Rotation* and *Delete*.

### 3.2.2.5 Saving options and formats

In order to allow the users to save the work done with SWift, multiple saving options have been developed. The saving form requires the user to enter two parameters: the title of his sign, and the save format. Each format yields different features.

By choosing the *image format*, the user is allowed to download an image which contains the sign present on the sign display. By choosing the *model format*, the user is allowed to download the sign as an XML-encoded file. Such file can also be re-opened by SWift, so that a previously composed file can be re-edited later. Choosing the *remote save* option, the association between the single glyphs (including their spatial positions) and the sign title, is saved into the database. This kind of saving carries many benefits for both SWift and the user: as the library of signs managed by the application grows, the automatic glyph hints, provided to the user during the composition process, are more accurate and consistent. Moreover, since the frequency of the glyphs is incremented for any sign they appear in, this procedure is useful for linguistic research.

### 3.2.2.6 Storyboard

The *storyboard* (Fig. 3.8) is the last feature, in chronological order, to be added to SWift. Unlike the features described in the previous sections, the design and development of the storyboard are part of the present work. The storyboard allows the user to compose not only single signs, but whole SL stories. A SL story is basically a series of signs, one after the other, arranged in a vertical order. The composition process starts as the user enters the title

of the story. After this, he/she is allowed to add signs to the story, using the classic interface for composing single signs. Following the instructions of the ISTC-CNR team, and the modern SignWriting specifications, we designed the storyboard so that the signs are stored in columns, from top to bottom, because deaf people naturally write with SignWriting in this way. The storyboard can store any number of signs, which are arranged in pages, and offers basic functions for the management of the single signs (create, edit, delete).

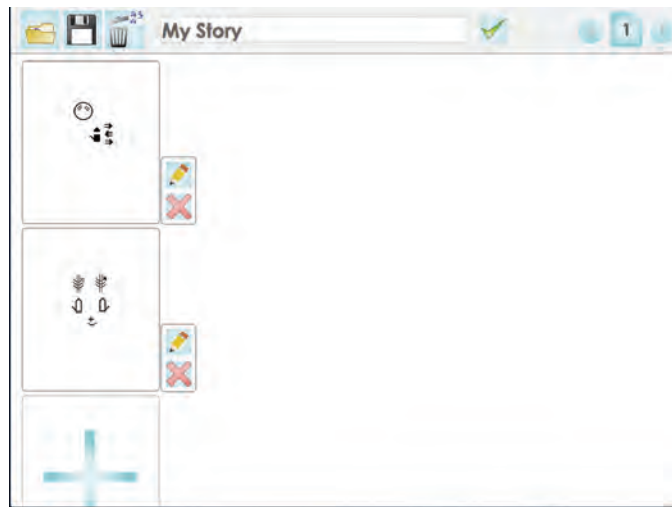


Figure 3.8: Storyboard: the story composition screen. The management buttons (open, save, delete) are visible in the title bar (top) and the signs already added are visible in the center of the screen, along with the buttons for their management (create, edit, delete).

As for the single signs, stories can be saved on the machine of the user, in two possible formats. Choosing the *document format*, the user can download a pdf document which contains the story. On the other hand, if the user wishes to modify the story at a later stage, he/she can save it using the *archive format*, which can be re-opened by SWift.

### 3.2.3 Usability test

#### 3.2.3.1 Methodology

The characteristics required from users to be included into the test were not easy to meet. As a matter of fact, in order to perform a reliable evaluation, users involved in the test were required to have at least a beginner-level knowledge of both SL and SignWriting. For such reason, the total number of participants was very small; the test was administered to a total of 8 people. The age of the participants ranged from 26 to 62 ( $\mu : 37.3$ ;  $\sigma : 11.7$ ), 5 (62.5%) of them being male. Most participants: 7 ( 87.5%) were deaf. Only 1 hearing person was found to present the adequate skills to participate to

the test. All the participants were very proficient in SL, their experience ranging from 5 to 51 years ( $\mu : 22.9$ ;  $\sigma : 15.0$ ), most of them, namely 6 (75%) were researchers in SL. The SignWriting expertise of the participants ranged from 0.5 to 6 years ( $\mu : 2.1$ ;  $\sigma : 1.7$ ); 6 (75%) of the participants already had experiences with digital SignWriting editors, in particular with SignMaker.

Given the particular situation, and in order to assess the usability of SWift, we adapted our testing approach to be viable for both SL and VL users. We chose a popular usability testing methodology, namely the Think-Aloud Protocol (TAP) (Lewis, 1982; Lewis & Rieman, 1993), and a widespread customizable questionnaire, namely the Questionnaire for User Interaction Satisfaction (QUIS) (Chin, Diehl, & Norman, 1998). In the following we describe how they were modified.

### Think-Aloud Protocol

Since deaf participants cannot actually “Think Aloud”, the TAP underwent a number of changes to include SL. The new version was named *Think by Signs*, since it can be used with deaf people who are able and willing to express their thoughts through SL. The TAP itself partly interrupts the attention flow of the user, since it engages cognitive resources along a different track. However, the positive aspect is the possibility to express one’s own impressions in real-time, without the possible bias due to the final outcome. In fact, a tool used for a successful task is often evaluated better than a tool used for a failed task, just for the positive result, and viceversa. On the other hand, we also administered a final questionnaire. Moreover, signing during different actions is typical of the way deaf people have to communicate while performing a task.

As explained in Chapter 1, deaf people also disclose their attitudes and sensations through a very high variability of SL non-manual components. Due to this characteristic, it is of paramount importance to capture the user’s face when recording a test session. Similar considerations already underlay the work by Roberts and Fels (Roberts & Fels, 2006), who performed a number of tests with deaf users adopting a TAP-based protocol. In particular, they asked participants to use a number of computer programs and to express, possibly by signs and through the translation of an interpreter, their impressions and final evaluation of such software. The authors also adopted a spatial setting in which two cameras operate to record the overall environment:

- **CAM 1** records the participant (from behind), the computer screen and the interpreter.
- **CAM 2** records the participant (front), and the investigator.

Analyzing two recordings at once and synchronizing the obtained data could prove a cumbersome task. For this reason, we decided to devise a novel spa-

tial setting which has the benefit of using one single camera, and to keep a synoptic view of anything happening in the environment which is worth being recorded. Fig. 3.9 shows our setting, which also takes advantage of a projector.

The orientation of the projector plays a key role, since, thanks to it, CAM1

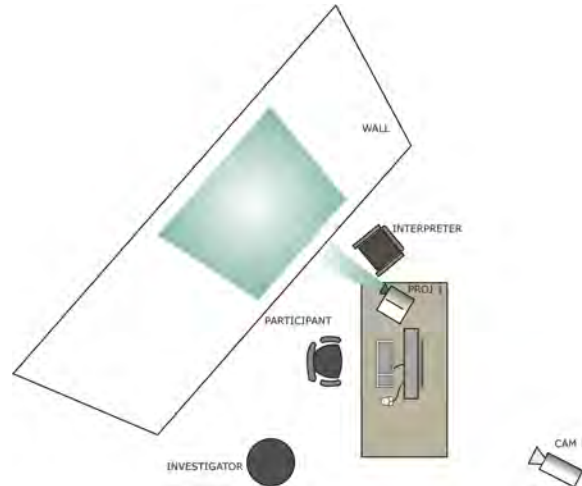


Figure 3.9: Spatial setting for the Think by Signs Test.

records a front view of the participant and of the interpreter, as well as the investigator, but also the computer screen, which is projected on the wall behind the participant, at an angle which is suitable to avoid disturbing occlusions between the elements of interest. Our different spatial configuration is not the only difference with (Roberts & Fels, 2006). As a matter of fact, after the definition of a proper spatial configuration, an even longer phase was necessary for the definition of modalities, elements and actors of the test itself. Such process was aimed at setting up a very general procedure which can be used with any software product.

We designed the Think by Signs Test to be composed by two moments. During the *welcome time*, which starts as soon as the participant sits down in front of the computer, the system displays a welcome screen containing a signed video (on the left part of the screen) and its VL translation (on the right part). Consistently with the rules of the TAP, the greeting sequence contains:

- A brief thanksgiving for the participation.
- A brief explanation about the structure of the test (e.g. “We will be giving you a list of tasks to perform”) and rules (e.g. “We recommend you to sign your thoughts as you perform the tasks”).
- A remainder about the purpose of the test, which is not conceived to test the skills of the participant, but rather to test the capabilities and the usability of the software; this should help the participant in feeling at ease with the test.

The *test time* represents the core part of the procedure. The participant is required to perform a list of tasks to test the functions of SWord and their usability. As stated by the rules of the TAP, during this phase the participant is asked to sign anything that comes to his/her mind. Given the possible high number of tasks (9 in our case, listed in Tab. 3.1), it is appropriate to alternate, when possible, simple jobs (like “Insert a glyph of your choice”) with complex ones (like “Compose the given sign”), to avoid tiring the participant. For the same reason, the participant might need a reminder about the task that he is currently carrying out. To address such need, we designed a “task list” to guide the participant. Since we deal with deaf people, we designed the task list to be available both in VL and SL.

TASK #	TASK
T1	Insert a glyph of your choice.
T2	Insert three given glyphs.
T3	Navigate to the home screen of SWord.
T4	Edit (rotate, delete) some of the glyphs within the whiteboard.
T5	Delete the whole content of the whiteboard.
T6	Compose the given sign.
T7	Handwrite 2 <i>ad hoc</i> glyphs.
T8	Save the sign in image format.
T9	Compose a sign of your choice.

Table 3.1: List of 9 tasks requested to the participants during the Think by Signs Test.

It is worth reminding that, notwithstanding the composition of the group involved into the test, our aim was to design a strategy suited for general cases of usability tests performed with mixed VL/SL groups of participants. Notice also that this is equivalent to a true bilingual setting.

Several options are available to create a bilingual task list. In the first place, the test designer should decide whether to delegate the SL inclusion to an electronic device (with signed videos illustrating each task) or to an interpreter. In the second place, consistently with the first choice, the specific role and responsibilities of the device/interpreter must be clearly defined. In our case, it was very important to avoid misunderstandings about the tasks requested to the participants. Such misunderstandings would bias the whole test because they would confuse the participant during the use of the tested system. In this context, we considered that the involvement of an interpreter usually makes the user feel more comfortable than a recorded explanation, due to the possibility to indirectly or directly ask question about the required activities. Of course, this also increases the probability of a correct understanding of the tasks. For this reason we chose an option involving the interpreter.

It is to notice that we prefer a clear division between the role of the inter-



preter and the role of the investigator. In other words, even if the investigator is able to fluently use SL, it is better to leave him completely focused on taking notes about anything interesting happening during the test. For this reason, we decided to involve a different person to act as interpreter. Given this, it is further necessary to define the way the interpreter has to act during the test. We think that the best option is to always provide an initial SL translation of the task. Therefore, at the beginning of each task, a card is presented to the participant. The card contains a simple, direct VL question which identifies the task (see Fig. 3.10). As the card is presented, the interpreter signs the question to the participant. Afterwards, the participant is allowed to ask questions to the investigator, using the interpreter as intermediary. The answer may follow only if it does not affect the outcome of the test (the participants were properly informed about this during the Welcome Time).

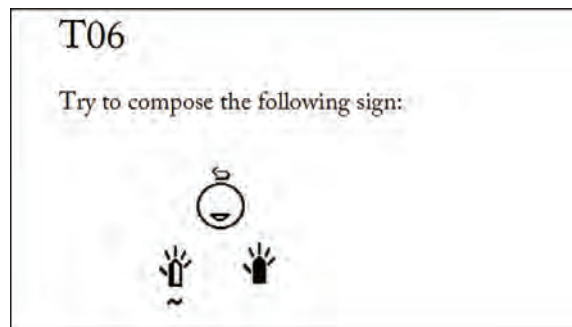


Figure 3.10: VL representation of task number 6 of the Think by Signs Test.

### Questionnaire for User Interaction Satisfaction

After the Think by Signs Test, the participant is left alone in the room, to avoid any conditioning, to fill in the *final satisfaction questionnaire*. The questionnaire was produced applying a number of changes to the QUIS (Chin et al., 1998). As stated by the authors, the QUIS “is designed to be configured according to the needs of each interface analysis by including only the sections that are of interest to the user” (Chin et al., 1998). For such reason, Section 1 (previous experience with the system), Section 8 (technical manuals and online help), Section 9 (on-line tutorials), Section 11 (internet access) and Section 12 (software installation) were removed because they were out of the scope of the test. Furthermore, a number of spare questions were removed from their respective categories, since they referred to features that SWift does not provide. Such questions are about the use of text messages/terminology and sound effects of the system. Tab.3.2 shows any change (and the corresponding rationale) we applied to the official version of QUIS (v.7) in order to build our final user satisfaction questionnaire.

Consistently with the first two phases of the test, the questionnaire was available both in VL and in SL. Our version of QUIS was implemented as a

#	SECTION NAME	ACTION	RATIONALE
N/A	Demographic questionnaire	KEEP	
1	Experience with the system	REMOVE	All participants were using the system for the first time.
2	Previous experience	KEEP	
3	General satisfaction	KEEP	
4	Screen	KEEP	
5	Terminology & System Info	CHANGE	Removed an item concerning text messages/terminology.
6	Learning	KEEP	
7	System Capabilities	KEEP	
8	Technical manuals and online help	REMOVE	Not available in SWift.
9	On-line tutorials	REMOVE	Not available in SWift.
10	Multimedia effects	CHANGE	Removed an item concerning the sound effects of the system.
11	Internet access	REMOVE	Not available in SWift.
12	Software installation	REMOVE	Not available in SWift.

*Table 3.2:* List of questions included in the QUIS employed for the test.

sequence of web pages. Each page presented a question, written in VL, accompanied by its corresponding SL video, showing an interpreter signing the question and its possible answers (see Fig. 3.11). Each page is conceived so that the user indicates its level of satisfaction on a visual scale, implemented via radio buttons (which are identified by very simple labels), ranging (in most cases) from total disappointment, to complete satisfaction. The user can also choose not to answer the question, choosing the “I don’t know” option. A number from 0 (bad) to 5 (good) was associated to each step of the satisfaction scale, in order to facilitate the analysis of the questionnaire. At the end of the test, the user could choose to activate a camera, built-in or placed near the computer, and to start signing any further comment.

In order to test the validity of some choices made during the design phase

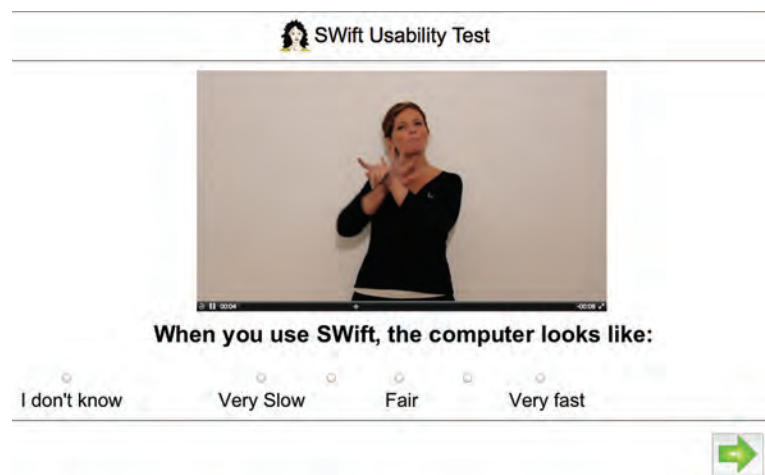


Figure 3.11: Web page showing a question of the QUIS employed for the test.

of the application, we decided to add 6 items to the test, tailored on specific aspects of SWift. Such items ranged from very specific, e.g. the usefulness of a particular User Interface (UI) component, to very general ones, such as the intuitiveness of the sign composition process. Such items were:

- Understanding the usage of the female character in the middle of the screen was: (Very hard - Very easy).
- Understanding the glyph search engine was: (Very hard - Very easy).
- Searching for the glyphs you needed was: (Very hard - Very easy).
- Using the glyphs you needed was: (Very hard - Very easy).
- Composing a whole sign was: (Very hard - Very easy).
- Using SWift, did you feel the need of a help function: (Yes - No).

### 3.2.3.2 Results

Each participant spent 7' to 30' ( $\mu : 18'$ ;  $\sigma : 7' 1''$ ) to complete the test. Each task took 0' 2'' to 14' ( $\mu : 2' 1''$ ;  $\sigma : 2' 44''$ ) to be carried out. To analyze the outcome of the test, we introduced the definition of *interesting event*. By this term we identify any noteworthy situation which occurred during the test, e.g. comments, errors etc. Each participant triggered a number of interesting events, ranging from 4 to 17 ( $\mu : 12.1$ ;  $\sigma : 3.9$ ). Such events help in identifying the areas of SWift which are worth improving. They have been analyzed and classified into five “Event Sets”:

- **S1 - Events related to interface appearance (39.8%)**: triggered by the lack of understanding of the UI components.  
**Example:** “The presence of graphics (instead of glyphs) in the home screen was misleading”.
- **S2 - Events related to application logic (24.7%)**: triggered by the lack of understanding of the application logic. **Example:** “Difficulties in understanding the usage of the glyph search engine”.
- **S3 - Events related to interaction modality (16.1%)**: triggered by the lack of understanding of the interaction rules of the application. **Example:** “Using drag instead of click to activate a button”.
- **S4 - Events related to glyph classification (11.8%)**: triggered by the arrangement of the glyphs and to their search criteria in the glyph search engine.  
**Example:** “Difficulties in locating the glyph for the hand-to-head contacts”.
- **S5 - Events related to bugs (7.5%)**: triggered by bugs in the application.  
**Example:** “Glyph rotation function not working properly”.

The events were not smoothly distributed over the tasks proposed to the participants, and were mainly focused on a restricted number of them. As a demonstration, Tab. 3.3 shows the event occurrence percentage (out of the number of total events recorded) for each of the 9 tasks. Two peaks were detected, one during T2 (34.5%) and another during T9 (23.6%), we will discuss such outcome in Section 3.2.3.3.

During each one of the tasks, different features and interaction modalities were tested. As a consequence, it is interesting to report the distribution of each event set for each task. Such distribution, shown in Tab. 3.4 varies dramatically from task to task.

In order to have a finer-grained analysis of SWift, we tracked the occurrence of each event. The count of the events is very high, some of them recurring only one or two times. Tab. 3.5 illustrates the 5 most recurring events during the test.

#	TASK	EVENT %
T1	Insert a glyph of your choice.	10.7%
T2	Insert three given glyphs.	34.5%
T3	Navigate to the home screen of SWift.	0%
T4	Edit (rotate, delete) some of the glyphs within the whiteboard.	1.1%
T5	Delete the whole content of the whiteboard.	5.4%
T6	Compose the given sign.	8.6%
T7	Handwrite 2 <i>ad hoc</i> glyphs.	12.9%
T8	Save the sign in image format.	3.2%
T9	Compose a sign of your choice.	23.6%

Table 3.3: Event occurrence percentage (out of the number of total events recorded) for each of the 9 tasks of the Think by Signs Test.

TASK	S1	S2	S3	S4	S5
Insert a glyph of your choice.	<b>70.0%</b>	20.0%	10.0%	0.0%	0.0%
Insert three given glyphs.	<b>50.0%</b>	33.3%	11.1%	2.8%	2.8%
Navigate to the home screen.	N/A	N/A	N/A	N/A	N/A
Edit some of the glyphs.	0.0%	0.0%	<b>100.0%</b>	0.0%	0.0%
Delete the whiteboard content.	<b>100.0%</b>	0.0%	0.0%	0.0%	0.0%
Compose the given sign.	25.0%	<b>37.5%</b>	12.5%	12.5%	12.5%
Handwrite 2 <i>ad hoc</i> glyphs.	0.0%	<b>57.11%</b>	42.8%	0.0%	0.0%
Save the sign composed.	<b>66.7%</b>	0.0%	0.0%	33.3%	0.0%
Compose a sign of your choice.	13.0%	8.7%	21.7%	<b>34.8%</b>	21.7%

Table 3.4: Percentage of occurrence of event sets for each task. The percentage of an event set for a given task is calculated out of the total events occurring during that given task. The most recurring event set is highlighted for each row. N/A is found only if no events were detected during a task.

#	EVENT	EVENT %
E02	The usage of the glyph search system is unclear. Consequent prolonged browsing.	20.6%
E03	Difficulties in locating the glyphs for the movement.	11.3%
E10	Presence of graphics (instead of glyphs) within the choose boxes is misleading.	9.3%
E01	Presence of graphics (instead of glyphs) in the home screen is misleading.	8.2%
E04	“Clear whiteboard” button is not well-recognized.	6.2%

Table 3.5: 5 most recurring events during SWift usability test.

The analysis of QUIS questionnaire indicated an overall satisfaction of the participants. The satisfaction was measured on a scale ranging from 0 (complete disappointment) to 5 (complete satisfaction). The average satisfaction level settled on 3.9 ( $\sigma$  : 1.0). More specifically, Tab. 3.6 shows the different satisfaction levels for each section of the QUIS.

QUIS SECTION	SATISFACTION ( $\mu$ )	SATISFACTION ( $\sigma$ )
Demographic questionnaire	N/A	N/A
Previous experience	N/A	N/A
General satisfaction	4.5	0.7
Screen	3.6	0.9
Terminology & System Info	3.8	1.0
Learning	4.2	0.8
System Capabilities	4.3	0.7
Multimedia effects	4.2	1.0
Custom SWift section	3.7	1.0

*Table 3.6:* Satisfaction level ( $\mu$  and  $\sigma$ ) for each section of the QUIS employed for the test.

All the participants accepted to leave their comments, after the QUIS. The comments were both positive and negative. More than half of the participants, mostly people with previous experience with other SignWriting digital editors, expressed their satisfaction with SWift, in particular with the simplicity of the composition process and with the UI of the application (words like “beautiful” and “warm” were used). The hint panel, and the associated glyph-suggestion function, was regarded as “one of the key feature of SWift” by a number of users. The negative comments, on the other hand, were very consistent with the distribution of the events of the Think by Signs Test shown in Tab. 3.5. The area which received most negative comments was the glyph search engine. The usage of such area remained unclear only for a very restricted number of participants, but many of them complained about being “overwhelmed” or “confused” by the big number of information (the glyph search results) provided by the system, and expressed the need of a graphical division between glyphs based on different prototypes. An important issue was the usage of graphical components, such as the puppet, in the home screen of SWift. In fact, a number of users expressed the need of actually “seeing glyphs” even on the home screen. Another recurring comment regarded the learning curve of the application: many participants reported that, after the first trial and error explorations which made them feel the interaction uncomfortable, finding glyphs and composing signs (and use the application in general) proved gradually easier.

Finally, no substantial performance/satisfaction difference was observed between deaf and hearing participants.

### 3.2.3.3 Discussion

Analyzing the distribution of the events for each task (Tab. 3.3), we observed a peak (34.5%) during T2, i.e. the second task in chronological order. Until that moment, the participant had only been asked to insert a glyph of his/her choice. The second task (“Insert three given glyphs”), however, forced the participant to deal extensively with the glyph search engine, which is a critical area in this application, as well as in any existing digital editor (e.g. SignMaker). Due to the large amount of data that need to be presented to the user, the search engine often needs some trial-and-error to be mastered. This generated a high number of early events, but after the first “impact” with the glyph search engine, events related to this area decreased (Tab. 3.4). Such decrease might most likely mean that the users learned to use it in a very short time. This hypothesis is supported by the results of the QUIS and by the comments of the participants. The second task, in terms of detected events, is T9, i.e. the last one. Since T9 required to the participant to compose a sign of his/her choice, the high number of events was most likely due to the high degree of freedom which characterizes this task. The participant was therefore able to follow his fantasy and take unpredictable paths, which sometimes lead him to search very particular glyphs. This is confirmed by the fact that most glyph classification related events occurred during this task (Tab. 3.4).

Analyzing the occurrence of each event (Tab. 3.5), we observed that most issues (20.6%) involve the glyph search engine, which needs some practice to be fully understood by the novel users. As an example we noticed that, although it was possible to check each choose box to refine the search results, most participants only checked one single choice, taking to a huge amount of candidate glyphs to browse. This is probably due to the fact that the possibility to make multiple choices is not sufficiently clear. The high number of comments about the glyph search engine, after the QUIS, confirm that this is an area that is worth improving. Another critical issue was the use of graphics (instead of glyphs) both in the home screen of the glyph menu (8.2%) and in some of its choose boxes (9.3%). Nevertheless, it is to notice that graphics (e.g. the picture of a hand) are present inside a choose box only if this gives access to a search area that cannot be symbolized with a single glyph, due to the number and the heterogeneity of the glyphs it contains. As an example, the glyphs for the movements of the hands are more than 14.000, so they cannot be summarized by one single glyph.

The results of the QUIS are almost completely in line with those of Think by Sign test. Tab 3.6 shows Section 4 (Screen) yielding the lowest level of satisfaction, more specifically, the average scores of item 4.2 “Highlighting on the screen simplifies task” ( $\mu$ : 3.3) and 4.3 “Organization of information on screen” ( $\mu$ : 2.8) are very low. This can be explained with the sensation, reported by the participants (see Section 3.2.3.2) of being “overwhelmed” by information when searching for a glyph. The custom section added to QUIS was meant to gather data about particular design choices. Two items,

in particular, received a very low score: item C-6 “Using SWift, did you feel the need of a help function” ( $\mu$ : 2.7) and item C-1 “Understanding the usage of the female character in the middle of the screen was [...]” ( $\mu$ : 3.3). This brought our attention to the necessity to build help functions and a tutorial for SWift, and, once more, to the necessity of improving the home screen of the application.

### 3.2.3.4 Limitations

The main limitation to the usability test of SWift is the restricted number of participants.

We also underlined that our aim was not to measure the user performance of SWift, in terms of sign composition time, but rather the ease of use of the application. It would have been worth to have a comparison in either one aspect or the other with SM. This should have required users with a similar experience in SignWriting and in the use of both applications. However, our sample included users that were well trained in the use of SM. Therefore, the past experience would have biased both performance and usability results.

### 3.2.4 After the test

A fair number of issues identified during the test were fixed. As an example, to fix the issue E10: “Presence of graphics (instead of glyphs) in the home screen is misleading”, we upgraded the puppet to show on mouseover, for each anatomical area, a sample of the most important glyphs it refers to. Fig. 3.12 shows the puppet interaction before the test (right) and after the test (left). The circle with the hand icon highlights the mouse pointer.



*Figure 3.12:* Difference between the puppet of V.1.00 (before the usability test) on the right and the puppet of V.1.01 (after the test) on the left; the circle with the hand icon highlights the mouse pointer.



## Chapter 4

# Introduction to SW-OGR

In the present work, we define SignWriting Optical Glyph Recognition (SW-OGR) as the electronic conversion of scanned images containing handwritten or printed SignWriting glyphs into machine-encoded SignWriting texts.

The present chapter explains the reasons to work for SW-OGR. Chapter 5 and 6 grant an in-depth view of our SW-OGR. Chapter 7 provides a performance analysis of the SW-OGR Engine.

### 4.1 Reasons to work for SW-OGR

The efforts of different teams around the world, including ours, produced a set of different SignWriting editors, as illustrated in Chapter 2. Despite their increasing capabilities in term of speed, usability, and reliability, such editors are still far from granting the user an interface able to emulate the simplicity of the handwriting. Actually, any software solution developed to support the production of SW documents relies heavily on Windows Icons Menu Pointers (WIMP) interfaces, both for accessing the application features and for the SignWriting production process itself. The problem is all but a theoretical one. Even dealing with word processors (Latin alphabet), the users often feel the higher complexity and the slower composition time with respect to handwritten text production, when no special formatting is needed. Given its huge number of glyphs, this especially holds for people using SignWriting, in particular for deaf people. In fact, they are far more accurate, fast, comfortable using the plain old paper-pencil approach rather than dealing with the (more or less) complex interaction styles of a digital editor.

Observing the intrinsic shortcomings of the present digital SignWriting editors (Chapter 2), we evaluated the possibility to design a new generation of SignWriting editing applications, able to partially overcome the concept of WIMP interface. The new tools are intended to lift the user of any burden related to clicking, dragging, searching, browsing on the UI during the SignWriting production process, and to implement an interaction style which is

as similar as possible to the paper-pencil approach that humans normally use when writing or drawing. Of course, since WIMP is currently the easiest, most common interface style in the world, it cannot be totally left behind, because it is necessary to access to the features of most applications. Nevertheless, our aim is to limit or dismiss the WIMP style during the SignWriting production process, which is the core part of any SignWriting editor. Our idea for the new generation of SignWriting digital editors is illustrated by the diagram in Fig. 4.1. We designed a OGR-powered SignWriting editor

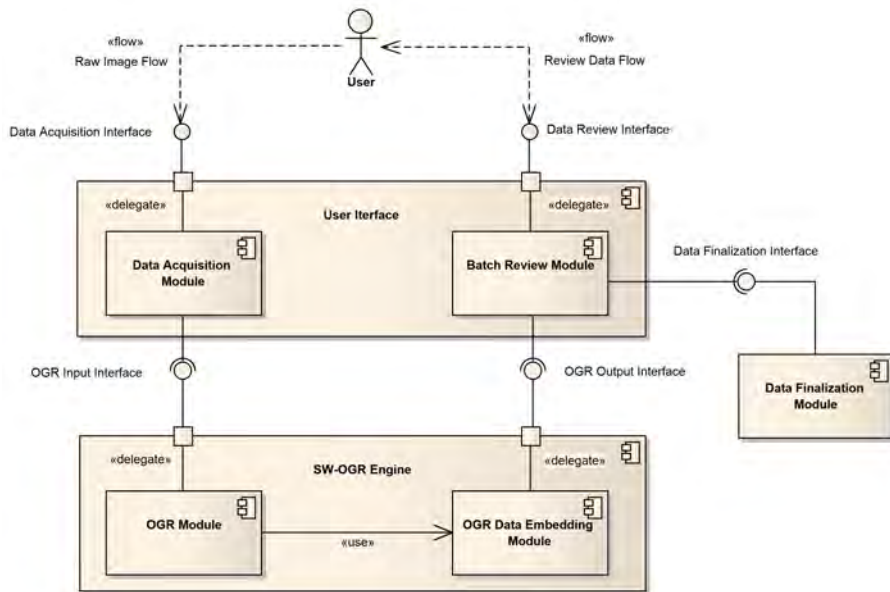


Figure 4.1: Component diagram for a new generation of SignWriting editors featuring a SW-OGR Engine.

composed by the following modules:

- The *Data Acquisition Module*, which is included within the *User Interface* (UI). The purpose of the UI is to provide the user with a simple interaction style for SignWriting composition, focusing on intuitiveness (or, better, *transparency*) and accuracy. An ideal setting to provide a paper-pencil-like tangible interaction style requires the usage of an additional hardware component: the graphic tablet. This module must also collect the data produced by the user (typically an image) and pass it to the SW-OGR module.
- The *SW-OGR Engine*, which is the core component of the application. Its purpose is to coordinate and control the recognition process. Two modules belong to the SW-OGR Engine:
  - The *OGR Module*, whose purpose is to provide a fast and accurate recognition of all (or most) glyphs composed by the user.

- The *OGR Data Embedding Module*, which is responsible for the creation of a result image embedded with the data produced by the OGR Module (it produces an image and an associated OGR data file, typically a SWML-encoded file<sup>1</sup>).
- The data from the SW-OGR Engine are sent back to the UI, and are shown within the *Review Module*, which also allows the user to make corrections and/or add other data. The work done to produce SWift could prove very useful during the development of this module, since their functionalities (glyph search and editing) are very similar.
- The *Data Finalization Module* which receives the user-reviewed OGR data from the UI. The purpose of this module is to save in the proper form (file, database, etc.) the data it receives.

The implementation of such an editor is the main reason why we decided to undertake the design and development of the SW-OGR Engine, since we think that it is bound to be the core component of a future SignWriting digital editor (as illustrated in Fig. 4.1).

As stated in the above list, the SW-OGR Engine is the component that is designed to carry out the recognition for the symbols within a SignWriting text. Before going further, however, it is important to introduce the definition of *online recognition* and *offline recognition*. On-line recognition systems recognize the text as it is being written (Khorsheed, 2002). The preferred input device is an electronic tablet with a stylus pen. The electronic tablet captures the (x,y) coordinate data of pen-tip movement, and an indication of pen-up and pen-down (Wakahara, Murase, & Odaka, 1992). The on-line recognition system has two major advantages: the high-recognition accuracy and the interaction (Khorsheed, 2002). The first advantage is that on-line recognition captures a character as a set of strokes, which are represented by a series of coordinate points. As a consequence, temporal information about the text is available to the recognition engine. The second advantage is that “it is very natural for the user to detect and correct unrecognized characters immediately by verifying the recognition results as they appear” (Khorsheed, 2002). On the other hand, on-line recognition is limited to recognizing handwritten text. It is evident, at this point, that the component diagram introduced in Fig. 4.1 depicts a system featuring online recognition. The off-line recognition system, on the contrary, recognizes the text after it has been written or typed (Khorsheed, 2002). The system may acquire the text using a video camera or a scanner. The latter is commonly used because it is more convenient, it introduces less noise into the imaging process. Typically, offline recognition is carried out by systems featuring high-speed scanners, featuring a high volume document feeder and high throughput Small Computer System Interface (SCSI) that can process a high number of pages per minute (see (Khorsheed, 2002) for more details).

---

<sup>1</sup>SignWriting Markup Language (SWML) is an XML-based format developed for the storage and processing of SignWriting texts and dictionaries

Providing online recognition features was not the only goal that we pursued. In fact, another motivation fueled our efforts while working on SW-OGR. Since the beginning of the project, we were aware of the presence (and of the considerable size) of a number of handwritten SignWriting corpora gathered from different communities around the world. Those corpora are an invaluable asset, and they could become even more useful if digitalized. We expect that SignWriting digitalization could allow a wider and faster diffusion of the information carried by the aforementioned corpora, as well as allow the linguistic research community to perform any kind of analysis on whole new SignWriting datasets.

Since the SW-OGR Engine provides very fast recognition routines for a large number of SignWriting glyphs, it can be employed for online (e.g. SignWriting editing) and offline (e.g. mass SignWriting corpora digitalization) processing. In the second case, the diagram in Fig. 4.1 can be easily adapted to avoid the constant presence of a human actor (Fig. 4.2). As illustrated in

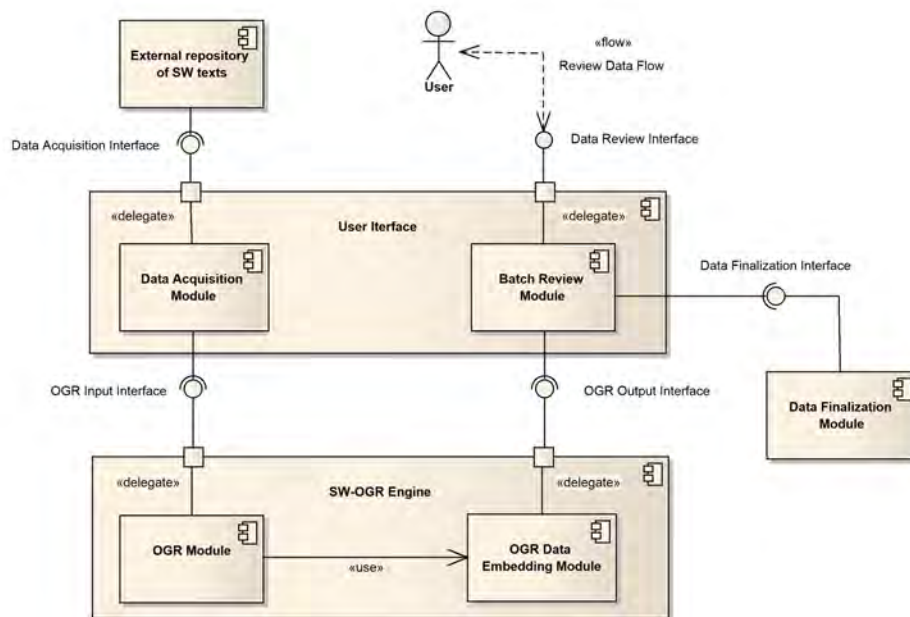


Figure 4.2: Component diagram for a new generation of mass SignWriting corpora digitalizer featuring a SW-OGR Engine.

Fig. 4.2, no human actor needs to interact with the system to provide input images, since the data flow is maintained by an *External Repository of SignWriting texts*. The images are handled, one by one, by the Data Acquisition Module.

Since the SW-OGR Engine works in the background of the application, and the image source does not affect its behavior, it presents no difference with the (above) case of the SignWriting editor.

Even in the case of offline recognition, it is not currently possible to completely avoid a human-in-the-loop approach. In fact, there may be errors in the recognition that only a SignWriting-proficient human being can correct. The *Batch Recognition Module*, however, is slightly different from its counterpart in Fig. 4.1 since its purpose is to present, on demand, the whole batch of OGR data produced, which undergoes a mass review process performed by one (or more) human supervisor. As usual, once the review is finished, the data is passed to the *Data Finalization Module* for the saving operations.

It is worth underlining at this point that the digitalization that we aim at is not the pure storage of digital SignWriting documents as images, which is actually easily accomplished by a simple scanning operation. We want to provide the same kind of integrated storage as SWift, where each sign is stored together with the list of component glyphs, in order to allow both the computation of linguistic statistics, and the enrichment of an effective recommendation/suggestion tool for the WIMP version of the editor. It is also worth noticing that, as in Latin alphabet, despite the glyphs allow to write any Sign Language, the co-occurrence frequencies underlying recommendation/suggestion depend on the specific language.

## 4.2 Related work on OCR

Apparently, the SW-OGR topic shares much in common with the Optical Character Recognition (OCR) techniques, which are well established within the computer vision literature. More specifically, we investigated the possible similarities between the SW-OGR and the modern OCR techniques employed to perform the recognition of Arabic, “Indian” (Hindi, Devangari, etc.), and Chinese handwritten characters. The comparison is motivated by the particular nature of such writing systems, which present rich visual features, so that they are more similar to the glyph-based system of SignWriting than any other alphabet. A further investigation was also performed in the field of Optical Music Recognition (OMR): a technique similar to OCR, specifically intended to automatically decode handwritten or printed musical scores and generate their digital representation.

Different writing systems pose challenges of varying difficulty to the recognition. As a consequence, before undertaking any implementation effort, it is mandatory to identify the structural features of the writing system in order to identify the most appropriate recognition strategies and techniques to be employed. Such analysis is the starting point of any computer science work related to OCR, and it must be performed by taking into account a number of factors. First of all, the arrangement of the text, i.e. whether it is written from left to right, as for Latin script, from right to left, as for Arabic script, from top to bottom, like the Mongolian and other Asian scripts, etc. Secondly, the symbols of the writing systems must be accurately analyzed, in order to have a global idea of their features and possible variations.

The recognition of Arabic script and the recognition of musical scores will be our main running examples through all the present section. Khorsheed (2002) reports a concise but effective analysis of the Arabic alphabet. The alphabet is composed by 28 basic letters, which consist of strokes and dots. Ten of them have one dot, three have two dots and two have three dots (Fig. 4.3, top). Dots can be above, in the middle or below the letter (Fig. 4.3, bottom). Besides this, a number of variation to the symbols should be taken into account, in fact “the shape of the letter is context sensitive, depending on its location within a word. A letter can have up to four different shapes: isolated, beginning connection from the left, middle connection from the left and right, and end connection from the right. Most of the letters can be connected from both sides, the right and the left. However, there are six letters which can be connected from one side only: the right” (Khorsheed, 2002). Moreover, some Arabic letters may have a zig-zag-like stroke called *Hamza*. This additional character can be only associated to a restricted number of letters, or isolated on the line. Another non-basic character is *Ta-Marbuta* which is a special form of one of the letters of the Arabic alphabet (namely *Ta'*), and it is always found at the end of the word.

Dot count	ن	ي	ش
Dot location	ن	ج	ب

Figure 4.3: Arab letters consist of strokes and dots. If they are present, dots can occur from 1 to 3 times (top). Moreover, dots can be above, in the middle or below the letter (bottom).

Another complex example is the set of characters in use in India, which can be used to represent 18 official (i.e. Indian constitution accepted) languages. Twelve different scripts are used to write these official languages. Analyzing such characters, it can be observed (Pal & Chaudhuri, 2004) that, apart from vowel and consonant symbols, called *basic characters*, there are *compound characters* which are formed by combining two or more basic characters. The shape of a compound character is usually more complex than the constituent basic characters. In some languages, a vowel following a consonant may take a modified shape, which depending on the vowel is placed to the left, right, top or bottom of the consonant. They are called *modified characters*. In general, there are about 300 character shapes in an Indian script (Chaudhuri & Pal, 1998).

Besides this, the appearance (and consequently the features) of the sym-

bols to be recognized may dramatically vary according to the *writing style*. Writing styles may be classified according to their complexity into three categories (Khorsheed, 2002):

- *Typewritten or machine-printed*: this is a computer-generated style, and it is the simplest among all styles because of the uniformity in writing a word.
- *Typeset*: this is normally used to print newspapers and books. Typeset style is slightly more difficult than the machine-printed style, because of the existence of possible *overlaps* and *ligatures*, which poses challenging problems. Ligatures occur when two or more letters overlap vertically and touch. By contrast, overlaps occur when two or more letters overlap vertically without touching.
- *Handwritten*: this is assumed to be the most difficult style because of possible variations and inaccuracies in the shape of the symbols.

Some other factors should be taken into account during the study of the writing system. One of the most important is the presence and the alignment of a *baseline*. A Baseline is an important characteristic of a large number of notations. In fact, it is a line (horizontal or vertical) that runs through the connected primitives of a text. In most notations, the baseline is assumed to have the maximum number of black pixels. This assumption, however, is not valid for all notations (i.e. it is not valid for East Asian ideograms). Moreover, if the script is skewed or handwritten, the baseline is not usually straight, and may only be estimated (Khorsheed, 2002). As anticipated, depending on the notation, the baseline can be both vertical or horizontal, and it can be placed in different positions with respect to the symbols. Fig. 4.4 shows different baseline alignments in different notations. From left to right, it is possible to identify the alphabetic baseline (Arabic, Latin, etc.), the hanging baseline (Indian scripts, etc.), and the ideographic baseline (East Asian ideograms, etc.).



*Figure 4.4:* Different baseline alignments in different notations. From left to right: the alphabetic baseline (Arabic, Latin, etc.), hanging baseline (Indian scripts, etc.), ideographic baseline (East Asian ideograms, etc.).

Once that all the defining features of the writing system to be recognized have been identified, the OCR procedure can be designed and implemented.

The study of the writing system is also a fundamental phase in OMR. In fact, music notation “is a kind of alphabet, shaped by a general consensus of opinion, used to express ways of interpreting a musical passage. It is the visual manifestation of interrelated properties of musical sound such as pitch, dynamics, time, and timbre” (Rebelo et al., 2012). Musical notation has a bi-dimensional structure, organized by the presence of the staff lines, and characterized by the existence of several combined symbols organized around the noteheads. The recognition of such symbols, especially when handwritten, poses challenges which are in all aspects as complex as the ones raised by the recognition of a written VL.

Before going further, some basic image processing definitions are needed in order to correctly understand how OGR/OMR procedures work. In image processing, we define an *object* as “a two- or three-dimensional thing that can appear in an image” (Parker, 2010). In the case of OCR, the term *object* can be further specified by defining it as a word, a part of a word, a symbol, or a part of symbol (Ahmed & Al-Ohali, 2000). In the case of OMR, on the other hand, the term *object* can be further specified by defining it as a staff, a clef, a beam, a stress symbol, an ornament symbol, a tie, a slur, a note, or a rest (see Rebelo et al. (2012) for further information).

A crude but functional definition of a *feature* is “something that can be measured within an image” (Parker, 2010). A feature is therefore a number or a set of numbers derived from a digital image. The idea underlying many approaches in image processing (pattern recognition above all), is that some objects belong to groups based on each of these measurements.

Finally, we define a *pattern* as “a combination of qualities (data) that form a characteristic arrangement” (Parker, 2010). Patterns can occur in numbers, in pixels, in sounds, or even in behavior. Patterns are important in computer vision because certain patterns of pixels represent specific objects in images. If those patterns can be detected, then it is an indication of the occurrence of that object in the picture. In the specific case of OCR/OMR, patterns are based on strokes, points and any feature that characterize the objects. It is important to remark that the definitions of object, feature and pattern are necessary to correctly understand how modern OCR/OMR procedures work. Regardless of the language and/or notation being recognized, most OCR/OMR approaches are implemented using a consolidated 6-step procedure (Ahmed & Al-Ohali, 2000). During a regular OCR/OMR session, a text undergoes the following steps:

- *Data acquisition* is the first step in the recognition system. The objective is to acquire the text and transform it into a raster image.
- The *pre-processing* step attempts to compensate for poor quality originals and/or poor quality scanning. This is typically achieved by reducing both noise and data variations.
- The *segmentation* step extracts the basic constituents from a given text. Such constituents typically correspond to the definition of objects



provided previously (Ahmed & Al-Ohali, 2000).

- The *feature extraction* step aims at capturing the essential characteristics of an object by exploiting the attributes which make it different from the other symbols.
- A major task after feature extraction is to perform the *classification* of the object into one of several categories.
- The *post-processing* step, if applicable, aims at using properties of natural languages to enhance the reliability of the recognition (Ahmed & Al-Ohali, 2000).

Each of the above step is achieved through the application of different techniques. Each step is further detailed (separately) in the following, through an overview of the most relevant techniques it relies on.

#### 4.2.1 Data acquisition

As anticipated earlier, the objective of this step is to acquire the text and transform it into a digital raster image. The image may be acquired in color mode, but the most common one is the grayscale mode (Parker, 2010). This is due to the lightweight features of the grayscale mode, and because, in most cases, a wider spectrum of colors is not needed.

The data acquisition plays an important role in shaping the whole recognition procedure. In fact, depending on the source of acquisition (scanner, tablet, camera, etc), the subsequent procedure may be an on-line or an off-line recognition (introduced in Section 4.1). An on-line OCR/OMR system recognizes handwritten text by capturing the pen positions in real time (the recognition procedure can exploit temporal information). An off-line OCR system recognizes an existing text (no temporal information available). In such a system, digital images of existing texts are produced by scanning them line by line or page by page. Afterwards these images are processed and analyzed (Ahmed & Al-Ohali, 2000).

#### 4.2.2 Pre-processing

Any OCR/OMR system depends upon both the original document quality and the registered image quality. The pre-processing stage attempts to compensate for poor quality originals and/or poor quality scanning. This is achieved by reducing both noise and data variations. All image acquisition processes are subject to noise of some type, therefore there is no ideal situation in which no noise is present. Noise can “neither be predicted nor measured accurately from a noisy image. Instead, noise may be characterized by its effect on the image” (Khorsheed, 2002). There are two types of noise: signal-independent noise and signal-dependent noise. Signal-independent noise adds a random set of grey levels, statistically independent

of the image data, to the pixels in the image. In signal-dependent noise, the noise value for each point  $p$  in the image is a function of the grey level of  $p$ . Investigating the state-of-the-art OCR (Khorsheed, 2002; Parker, 2010) and OMR (Göcke, 2003; Rebelo et al., 2012) techniques, it is possible to observe that coping with noise, and image pre-processing solutions in general, are the same in both cases.

*Smoothing* is one of the most common ways to cope with noise within an image. The smoothing process typically reduces the noise in an image using a wide range of operations. *Opening* and *closing* are among the operations which are frequently used to perform smoothing. “Opening opens small gaps or spaces between touching objects in an image; this will break narrow isthmuses and eliminate small islands. In contrast, Closing fills small gaps in an image; this will eliminate small holes on the contour” (Khorsheed, 2002). Please notice that both opening and closing apply the same basic morphology operations, namely, *dilation* and *erosion*, but in the opposite order. Opening applies an erosion operation immediately followed by a dilation operation. Closing applies a dilation operation immediately followed by an erosion operation, see Khorsheed (2002); Parker (2010) for more details.

Another approach to reducing noise (more specifically, salt-and-pepper noise) is by applying the *median filter*. This is performed by passing a small window through all pixels in the image. The pixel in the center is replaced by the median value of all the pixels in the region.

The presence of noise is not the only issue addressed during the pre-processing step. In fact, scanning/writing a document so that text lines are within about three degrees of the true horizontal is acceptable. However, for different reasons related both to the acquisition device and to the actual content of the original image, this is not always possible. As a consequence, one of the first steps in attempting to read this document is to estimate the orientation angle, i.e. the *skew angle*, of the text lines. This process is referred to as *skew detection*, and the process of rotating the document with the skew angle, in the opposite direction, is named *skew correction*. The common, and perhaps the most efficient, approach to estimate the skew angle is to use the *Hough Transform* (Hough, 1959), which is a method for detecting straight lines in a raster image. Once the image has been pre-processed, it is ready to undergo segmentation.

### 4.2.3 Segmentation

A document image is a visual representation of a printed page. Typically, this page consists of blocks of text (which can possibly be interspersed with tables and figures). During the segmentation process, the basic constituents are extracted from a given page. In Arabic OCR, for example, the basic constituents can either be a word, the complete shape of a character or the partial shape of a character (Ahmed & Al-Ohali, 2000). In most writing systems, methods of deriving the blocks can take advantage of the fact that the structural elements of a document are generally laid down in rectangular

blocks aligned parallel to the horizontal and vertical axes of the page. The document decomposition and structural analysis task can be divided into three phases (Srihari, Lam, Govindaraju, Srihari, & Hull, 1992):

- During phase one, i.e. block segmentation, the document is decomposed into several rectangular blocks. Each block is a homogeneous entity which may contain a text, an image, a diagram or a table.
- During phase two, i.e. block classification, each block is assigned a label (title, regular text, picture, table, etc.) using properties of individual blocks from phase one.
- During phase three, the focus is mainly on text blocks, which undergo a logical grouping and ordering.

The first challenges that almost any segmentation procedure must face is the separation of lines (or columns) and the segmentation of words and sub-words. A very common approach (Fehri & Ahmed, 1994) to identify the lines in the text implies computing the horizontal projection of each row within the image; of course, vertical projection is used to identify columns, if the writing system has a vertical arrangement. After this, the procedure looks for rows whose projection is equal to zero (no foreground pixel along the row), then consider that every text line is situated between two blocks of zero density pixel lines. This method is enhanced by identifying the lines of pixels that have the largest density in the text. Words and sub-words are determined by inspecting the projection along the remaining Cartesian axis, i.e. vertical, if the text has an horizontal organization, and viceversa. An average threshold value computed from all vertical gaps is used to determine whether a spacing is an inter-word spacing or an intra-word spacing (Altuwaijri & Bayoumi, 1994).

Another attempt at decomposing text into words, is based on the connected components of that script. This method can solve the difficulty of segmenting handwritten script, where a clear cut between two consecutive text line may be not possible to find (Abuhaiba, Holt, & Datta, 1998).

Regarding the Arabic notation, however, the classical method for identifying text lines in an Arabic text image is to use a fixed threshold to separate the pairs of consecutive lines (Amin & Masini, 1986; Khorsheed, 2002). This threshold is obtained using the distances between various baselines of the text. The median of different distance values is an appropriate selection.

In OMR, on the other hand, the actual segmentation of the text is typically performed in the phases:

- During phase one, the staff lines of the page are detected and removed.
- During phase two, the musical blocks are located and isolated for feature extraction and classification.

Staff line detection and removal are “fundamental stages in many optical music recognition systems. The reason to detect and remove the staff lines

lies on the need to isolate the musical symbols for a more efficient and correct detection of each symbol present in the score” (Rebelo et al., 2012). The simplest approach consists of finding local maxima on the horizontal projection of the black pixels of the image (Fujinaga, 2005). Assuming straight and horizontal lines, these local maxima represent line positions. Several horizontal projections can be made with different image rotation angles, keeping the image where the local maximum is higher. This eliminates the assumption that the lines are always horizontal. The Hough Transform is also used widely in a number of approaches (Miyao & Nakano, 1995), in order to detect staff lines. Other approaches (Fujinaga, 2005) incorporate a set of image processing techniques in the staff detection algorithm, including run-length encoding, connected-component analysis, and projections. After applying the run-length encoding to find the thickness of staff lines and the space between the staff lines, any vertical black run that is more than twice the staff line height is removed from the original. Then, the connected components are scanned in order to eliminate any component whose width is less than the staff space height. After a global de-skewing, taller components, such as slurs and dynamic wedges are removed.

The extraction of music symbols is the operation following the staff line detection and removal. The segmentation process consists of “locating and isolating the musical objects in order to identify them. In this stage, the major problems in obtaining individual meaningful objects are caused by printing and digitalization, as well as paper degradation over time. The complexity of this operation concerns not only the distortions inherent to staff lines, but also broken and overlapping symbols, differences in sizes and shapes and zones of high density of symbols” (Rebelo et al., 2012). The most usual approach for symbol segmentation is a hierarchical decomposition of the music image. A music sheet is first analyzed and split by staves and then the elementary graphic symbols are extracted: noteheads, rests, dots, stems, flags, etc. (Choudhury, Droetboom, Dilauro, Fujinaga, & Harrington, 2000; Fujinaga, 2005; Miyao & Nakano, 1995).

#### 4.2.4 Feature extraction

In order to carry out the recognition, OCR/OMR procedures evaluate the features of the objects. As a matter of fact, such features are the properties that make one object different from another. Therefore, their definition, extraction and classification are of paramount importance to implement any OCR/OMR procedure. The selected set of features should be a small set whose values efficiently discriminate between patterns of different classes, but are similar for patterns within the same class. The feature extraction step is closely related to classification because the type of extracted features must match what the classifier expects (Al-Badr & Mahmoud, 1995). Feature types can be categorized as *structural features* and *statistical features*.

### Structural features

Structural features describe a pattern in terms of its topology and geometry by giving its global and local properties (Gonzalez & Woods, 2011; Parker, 2010). Such type of features can highly tolerate distortions and variations in writing styles, but extracting them from images is not always easy. Moreover, the structural features used in OCR literature depend on the kind of pattern to be classified (Al-Badr & Mahmoud, 1995). As a consequence, this aspect of the OCR is highly depended on the writing system which is to be recognized. In other words, OCR procedures for different writing systems extract different features. Within the same OCR procedure, structural features can be organized in different, independent divisions, each one extracting a different set of features (Hassibi, 1994). Very often, based on the preliminary features, an object may be assigned to a certain group where further feature extraction is carried out (Amin, 1998; Gonzalez & Woods, 2011).

Resuming our running example, the features extracted from Arabic symbols include strokes and bays in various directions, endpoints, intersection of line segments, loops, stroke positions relative to the baseline, dots and their positions relative to the baseline, and zigzag (Al-Badr & Mahmoud, 1995; Khorsheed, 2002). Features can also be extracted from component curves and strokes of a symbol. Such features include: the direction of curvature (e.g., clockwise), the direction, slope, and length of strokes (Ahmed & Al-Ohali, 2000), the length of a contour segment, the distance between the start and end point of the contour projected on the X- and y-axis, and the difference in curvature between the start and end points (Margner & El Abed, 2007). Sometimes the pattern is divided into several zones and several types of geometric features in each zone are registered and counted, with some features constrained to specific zones (Al-Badr & Mahmoud, 1995). Those features include the number of concavities, holes, cross points, loops, and dots; the number and length of the contour segments; the zone with maximum (minimum) number of pixels; and concavities in the four major directions (Parvez & Mahmoud, 2013).

Similar techniques are also employed in music score recognition: a number of authors (Bellini, Bruno, & Nesi, 2001; Fujinaga, 2005) have chosen to apply projections to detect primitive symbols, and to extract features from the projection profiles. Another approach for the extraction of structural features is based on the construction of graphs for each symbol. The symbols are isolated and analyzed by using a region growing method and thinning (Rebelo et al., 2012). Other approaches (Choudhury et al., 2000) propose the extraction of symbol features, such as width, height, area, number of holes and low-order central moments; slightly different approaches (Taubman, 2005) extract standard moments, centralized moments, normalized moments and Hu moments.

### Statistical features

Statistical features are derived from the statistical distribution of pixels and describe the characteristic measurements of a pattern. Generally speaking, statistical features are easy to extract. Nonetheless, they may be misleading, due to a fraction of noise brought forth haphazardly according to the binarization process.

Statistical features typically include zoning (Bazzi, Schwartz, & Makhoul, 1999), which considers the density distribution of character pixels, characteristic loci (Abdelazim, Mousa, Saleh, & Hashish, 1990), which counts the one and the zero segments and the length of each segment, the ratio of pixel distribution between two parts of the image (Bousslama, 1996) and moments (Sanossian, 1996).

Regarding the OCR for Arabic script, the text line was partitioned by some researchers (Bazzi et al., 1999; Makhoul, Schwartz, Lapre, & Bazzi, 1998) into a sequence of narrow vertical overlapping frames, with a width that was a small fraction of the height of the line. Each frame was divided into 20 equal overlapping cells. Features extracted from each cell were: the intensity of a cell, the vertical and horizontal derivative of intensity, and the local slope and correlation across a window of two cells. The frame could have a one-pixel width, as in Abdelazim et al. (1990). The system extracted a feature vector from each column of pixels in the word image, where each feature represented the length of black/white pixel run. Moment invariants refer to certain functions of moments (Jain, 1989), which are invariant to geometric transformations such as translation, scaling and rotation (Khorshed, 2002). It is important to notice, however, that moment invariants are sensitive to any change and multi-font recognition (Al-Badr & Mahmoud, 1995; Khorshed, 2002).

#### 4.2.5 Classification

As previously anticipated, a major task after feature extraction is to classify the object into one of several categories. In computer science, more specifically in machine learning, a *classifier* is a function that takes the values of various features (independent variables) in an example (the set of values of the independent variables) and predicts the *class* that that example belongs to (the dependent variable), on the basis of a *training set* of data containing observations (or instances) whose class membership is known.

#### Minimum distance classifier

There are a number of various classification techniques applied in text recognition. One of the most popular is the *Minimum distance classifier*. Given  $K$  different classes, where each class is characterized by a feature vector prototype, the problem is to assign an input feature vector to one of these classes according to a predefined discriminant function. The features can

typically be geometrical, statistical, or structural (see Khorsheed (2002) for a complete bibliography).

### Decision tree classifier

Another relevant classifier, widely used in OCR (Khorsheed, 2002), is the *decision tree classifier*. This classifier splits the  $N$  dimensional feature space into unique regions by means of a sequential method. More specifically, decision tree learning uses a decision tree as a predictive model which maps observations about an object to conclusions about the object's target value. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

The decision algorithm is typically designed such that every class need not be tested to arrive at a decision. This becomes advantageous when the number of classes is very large. In the specific case of OCR, classification here can be a two-step process (Goraine & Usher, 1994; Khorsheed, 2002). In the first step, an input character is assigned to one of the main groups according to some syntactic rules. Then, and relative to a more detailed feature vector, the input character is matched with one of the group members.

### Statistical classifiers

*Statistical classifiers* assume that different classes and the feature vector have an underlying joint probability. One approach is to use the Bayes classifier (Devroye, Györfi, & Lugosi, 1996). The Bayes classifier minimizes the total average loss in assigning an unknown pattern to one of the possible classes. The probability density function can be cumulative, therefore at the end, the assignment is to that class with majority samples.

Statistical classifiers are widely used in Arabic OCR (see Khorsheed (2002) for a complete bibliography). Adopting statistical classifiers. Al-Badr and Haralick (1996) implemented a three-step recognition process: first they found instances of a set of shape primitives on a text image; then they took the detected primitives of a word and hypothesized a number of alternative strings as the recognition of the word. The choice would be on the one with the maximum *a posteriori* probability. Finally, the probability of a match was computed between the symbol model and the word image.

Hidden Markov Models are statistical models which have been found extremely efficient for a wide spectrum of applications, especially speech processing (Gales & Young, 2007). This success has motivated researchers to implement HMMs in character recognition. Abdelazim et al. (1990) first applied HMMs to recognize Hindi numerals. Each numeral was represented with a separate HMM. The observation sequence was passed to all the ten models, and assigned to the numeral with the highest model probability of the observation sequence.

HMMs are also employed within a number of novel OMR approaches (Pu-

gin, Ashley, & Fujinaga, 2007.) which avoid the segmentation phase before the feature extraction and classification phase. In fact, such methods simultaneously perform segmentation and recognition. The extraction of features directly from the image frames (rather than from segmented objects) has advantages. Particularly, it avoids the need to segment and track the objects of interest, a process with a high degree of difficulty and prone to errors (Rebelo et al., 2012).

### Neural networks classifiers

Finally, *neural networks classifiers* are a further example of technique applied to OCR. Actually, OCR is one of the most successful applications that has been proposed for Neural Network (NN). A NN is a non-linear system which may be characterized according to a particular network topology (G. Zhang, 2000). This topology is decided by the characteristics of the neurons and the learning methodology. NNs can simply cluster the feature vectors in the feature space, or they can integrate feature extraction and classification stages by classifying characters directly from images.

There are three main advantages behind implementing NNs in OCR: NNs have faster development times; they have an ability to automatically take into account the peculiarities of different writing/printing styles; and they can be run on parallel processors. Generally speaking, the common architecture of NNs used in Arabic OCR is a network with three layers: input, hidden and output. The number of nodes in the input layer varies according to the dimension of the feature vector or the segment image size. The number of nodes in the hidden layer govern the variance of samples that can be correctly recognized by this NN. On the other hand, introducing a new shape to the NN requires that the network be retrained, or even worse, that the network be trained to a different architecture. Intensive work can be found on the subject of Arabic OCR using neural networks (see Khorsheed (2002) for a complete bibliography). NNs were also applied to recognize Arabic words on-line Khorsheed (2002).

Regarding Indian OCR, the viability of NNs has been explored only for 2 out of 12 scripts, namely Telgu (Sukhaswami, Seetharamulu, & Pujari, 1995) and Oriya scripts (Mohanti, 1998). About this latter script, Mohanti (1998) proposed a system to recognize alphabets of Oriya script. The inputs pixels are fed to the neurons in the Kohonen layer where the neurons determine the output according to a weighted sum formula. The character is classified according to the largest output obtained from the neuron. The authors made experiment only on five Oriya characters and hence the reliability of the system is not fully established.



### 4.2.6 Post-processing

The final stage in the recognition process is post-processing. One of the objectives of post-processing is to improve word recognition rate (as opposed to character recognition rate). Post-processing is often implemented as a set of techniques that rely on character frequencies, lexicons, and other contextual information. As classification, sometimes, produces a set of possible solutions instead of a unique solution, post-processing is responsible for selecting the right solution using higher level information that is not available to the classifier. Post-processing also uses that higher level information to check the correctness of the solutions returned by the classifier. The most common post-processing operations are spell checking and correction. Spell checking can be as simple as looking up words in a lexicon. To improve the speed of lookup, a lexicon is sometimes represented as a tree (Khorshed, 2002). When spell checking fails, Amin and Mari (Amin & Masini, 1986) use the Viterbi Algorithm to find alternate words whose characters, with a high probability, can be interchanged with the original ones, using a Hidden Markov model for each word. During the post-processing stage in a restricted number of OMR approaches, the role of the user is of cardinal importance in the recognition process. In fact, “an automatic OMR system capable of recognizing handwritten music scores with high robustness and precision seems to be a goal difficult to achieve” (Rebelo et al., 2012). Hence, interactive OMR systems are a realistic and practical solution to this problem. MacMillan, Droettboom, and Fujinaga (2002) adopted a learning-based approach in which the process improves its results through experts users. The same idea of active learning was suggested by Fujinaga (2005) in which “a method to learn new music symbols and handwritten music notations based on the combination of a k-nearest neighbor classifier with a genetic algorithm was proposed” (Rebelo et al., 2012).

## 4.3 SW-OGR issues at a glance

We decided to implement the 6-steps procedure which is distinctive of the modern OCR techniques within our SW-OGR Engine (our implementation is covered in Chapter 6). Actually, the early and final steps of the OCR, i.e. acquisition, segmentation and post-processing, present little or no difference as the language to be recognized varies. In fact, the techniques employed for the implementation of such steps are very general purpose ones, i.e. they are not language-specific (Parker, 2010). Therefore, we decided to adopt the traditional OCR approach for the acquisition, pre-processing and segmentation steps, using a number of consolidated techniques in computer vision literature (an in-depth account of the employed techniques is covered in Chapter 6).

On the contrary, the core steps of the procedure, i.e. feature extraction and classification, typically make large use of pattern recognition and machine-learning techniques (as explained in Section 4.2), that aim at assessing both

the quantitative (statistical, numerical) and qualitative (structural, topological) features within the texts.

Unfortunately, we could not implement any pattern recognition and machine learning approach within the SW-OGR Engine, and we were forced to devise our own feature extraction and classification algorithms. The reason for this choice is the very particular nature of SignWriting. More specifically, the ISWA contains a number of symbols which is in the order of the tens of thousands. Moreover, since handwriting is often inaccurate, each symbol can be drawn in a number (usually in the order of tens) of different ways, as in any handwritten notation. Last but not least, some details, e.g. the number of eyelashes on an eye, can be chosen according to the writer preference, since the only relevant information is the presence of an open eye with a certain expression. As a consequence, it is evident that any training to enable a machine learning approach would take an unreasonable amount of time, even for a subset of the ISWA, except for the basic shapes which make up only the core part of the glyphs. And, actually, we used them for this aim.

Another reason makes the application of pattern recognition techniques very difficult, i.e. the total lack of rules regulating the number, type and position of the glyphs within a SignWriting sign. In fact, the composition freedom, which is one of the features which make SignWriting a very handy tool for everyday SL writing, also makes the modeling of the notation very difficult. For different reasons, in fact, it is not rare to see signs written with 2 or more head symbols, or 3 or more hand configurations (to express, for example, a specific movement). No study has been performed so far to perform a statistical modeling about the type, number and position of the glyphs within a SignWriting sign. This is one of the issues that our framework might help to address. Moreover, the question also presents practical issues, since any rigid modeling could basically be perceived by SignWriting users as a “limit” to the distinctive freedom of composition boasted by the writing system.

The lack of statistical analyses, and previous studies on SignWriting, in combination with the other two factor mentioned above, contributed in making statistical feature extraction and classification not practically viable. For the above reasons, we investigated the possibility to achieve SW-OGR by working with three main sources of information:

- Structural (geometric and topological) features of the glyphs.
- Topological relationships among the glyphs.
- Context-dependent information, such as the knowledge of the organization of the ISWA.

Finally, for the reasons already introduced in the present section, more specifically the lack of statistical analyses on SignWriting compositions, and the high level of freedom featured by the system, we could not perform the post-processing step automatically. As mentioned in this chapter, one

---

of the objectives of post-processing is to improve the recognition rate of words (signs, in our case). This step is typically implemented using a set of techniques that rely on character frequencies, lexicons, and other contextual information. At present, no such study has been carried out for SignWriting compositions, so our post-processing step is performed with the aid of a human actor (as detailed in Section 4.1). As a consequence, the post-processing step is not managed by the SW-OGR Engine.

A theoretical study (illustrated in the present Section) of the SW-OGR topic was necessary prior to any engineering effort. Once the main algorithms and entities involved in the recognition were defined (Chapter 5), the implementation (Chapter 6) and testing (Chapter 7) phases have taken place.



## Chapter 5

# Coding system of SW-OGR

### 5.1 SignWriting datasets

Before starting any work for the SW-OGR Engine, it was necessary to gather a comprehensive dataset of SignWriting texts. The present section provides a sample of each dataset we used. We made sure to obtain datasets containing both handwritten and printed symbols, in order to work on a wider variety of symbols.

The first dataset we obtained, in chronological order, is the *Pear Story dataset* (DS-PEAR). DS-PEAR is composed by 58 pages containing LIS narrative texts produced by the signers of the SLDS laboratory via the presentation of Chafe's (Chafe, 1980) *Pear Story* film. Each text includes a transcription from a video-recording and from a face-to-face signed production. DS-PEAR was extracted from Bianchini (2012), a sample of the dataset is shown in Fig. 5.1.

Another handwritten dataset is the *Poitiers dataset* (DS-POITIERS). DS-POITIERS was provided by C.S. Bianchini, associate professor in linguistics at the Université de Poitiers. The dataset is composed by 156 pages containing LSF texts produced during different exercises by her students in linguistics. A sample of the dataset is shown in Fig. 5.2.

SW-PEAR and SW-POITIERS are our main, original datasets. Sometimes, however, especially during the development and the testing phase, it is useful to have an utility dataset, built to test single features of the recognition engine. For this reason, we created a *Development dataset* (DS-DEV), whose texts contain symbols belonging to the same SignWriting category, or different variations of the same symbol. As an example, in order to test the recognition of the Hands Configurations area, we built a text containing a high number of possible rotations, plane and hand variation of the same configuration. The text is shown in Fig. 5.3. Even though the differences are difficult to spot, please notice that DS-DEV was produced by the SW-OGR development team, rather than SignWriting experts. Finally, as already mentioned, we gathered a number of printed SignWriting texts in order to assemble the *Printed dataset* (DS-PRINTED). Part of such texts were composed by the

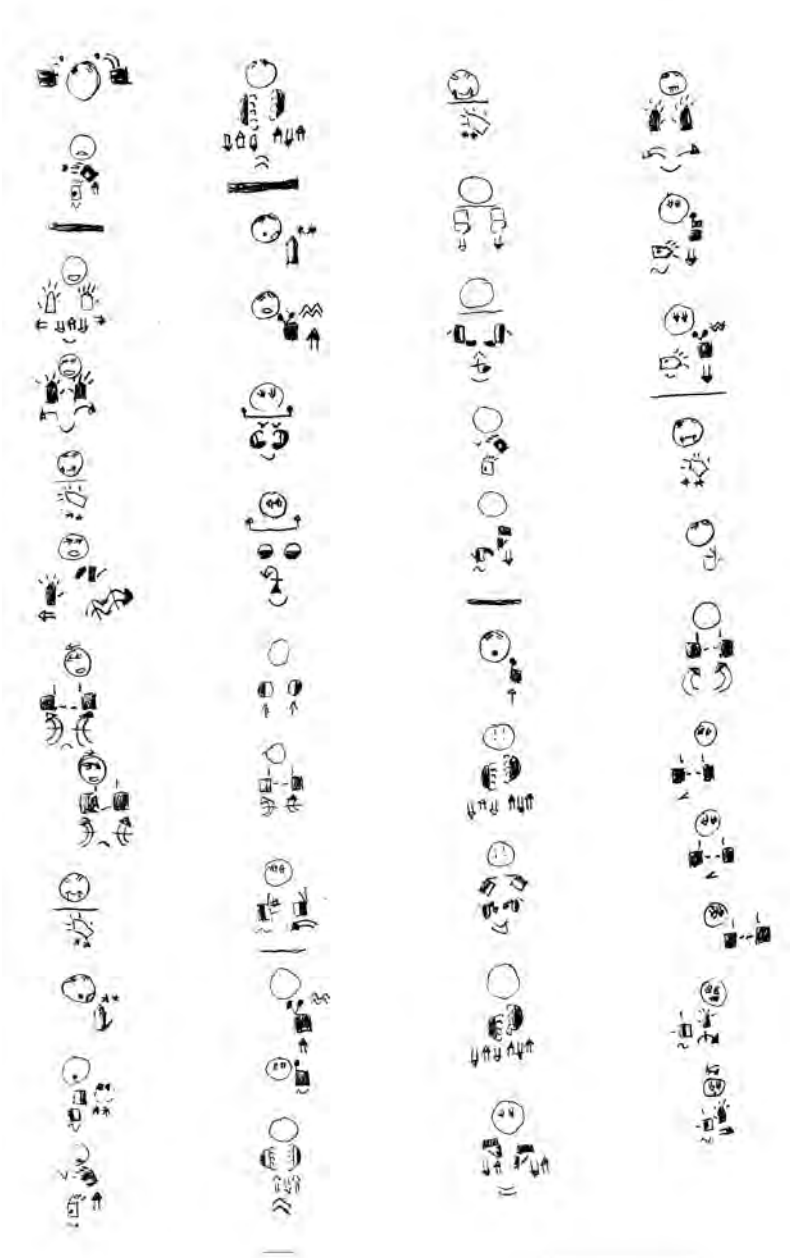


Figure 5.1: Sample of the DS-PEAR dataset.

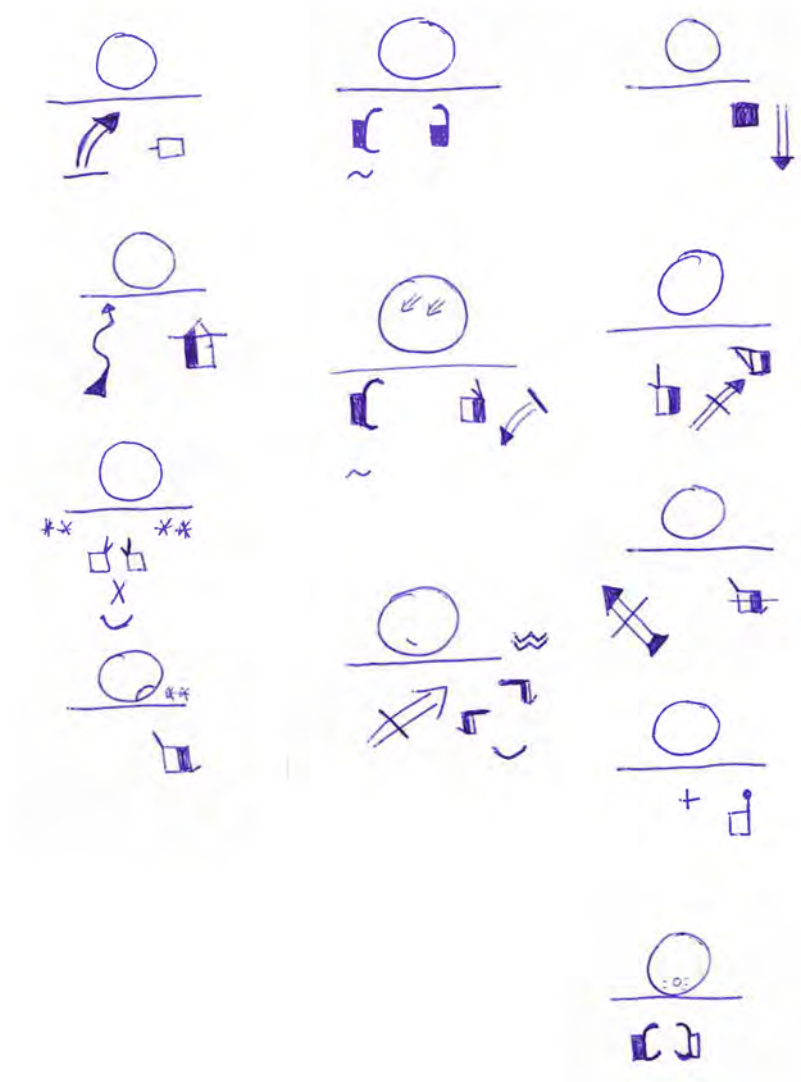


Figure 5.2: Sample of the DS-POITIERS dataset.

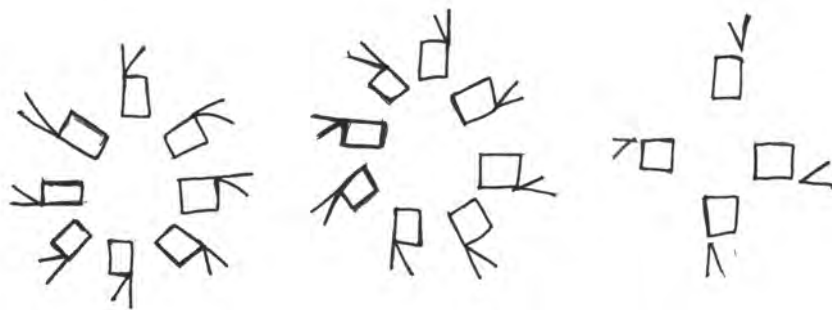


Figure 5.3: Sample of the DS-DEV dataset.

SW-OGR development team using SWift, while another part was gathered from the SignWriting Website (Sutton, 1996). The purpose of such dataset is the same as DS-DEV, i.e. an utility dataset which can be exploited in multiple occasions during the development of the application. Moreover, printed glyphs are often fairly different from handwritten glyphs, so DS-PRINTED was also useful to test the robustness of our recognition procedure.

## 5.2 Core recognition elements

The full understanding of the terminology that we will be using in the following chapters is of critical importance. Therefore, it is now worth introducing a number of concepts that will be very important during the description of the design and development of the SW-OGR Engine. Any keyword introduced in this section is also available in the Glossary.

In the beginning of our work, we observed a number of SignWriting handwritten texts from a conceptual point of view. Our aim was to perform a top-down analysis of the graphical elements we would have to work with within the image. *In primis*, the first element the SW-OGR has to deal with is the image itself, as a whole. Such image contains handwritten or printed SignWriting glyphs. We define such input image as a *text* (see Fig. 5.4).

*A text is an image containing handwritten or printed SignWriting glyphs.*

According to the visual organization of the text, it is possible to divide the image in sections, i.e. rows (horizontal organization) or columns (vertical organization). As illustrated in Chapter 2, SignWriting is naturally written using a vertical visual organization, and all the datasets we used to test the SW-OGR Engine follow such organization. It is not impossible, however, for a text to follow a horizontal organization. Anyway, a text typically contains one or more of such sections, each one of them is in turn composed by a set of symbols. We define these sections as *slices* (see Fig. 5.5).

*A slice is a section of a text. If the text follows a horizontal visual organization, a slice is a row within the text. Otherwise, if the text follows a vertical visual organization, a slice is a column within the text.*

Each slice is composed by a set of connected components, that we define as *frags* (see Fig. 5.6).

*A frag is a 8-connected foreground component within a slice.*



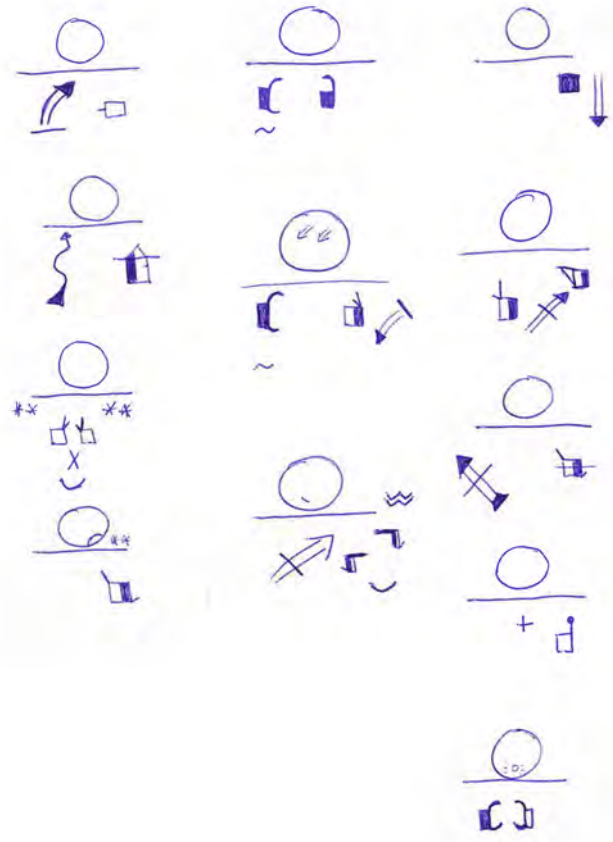


Figure 5.4: An example of text belonging to the DS-POITIERS dataset.



Figure 5.5: An example of slice (courtesy of C.S. Bianchini). This is the first slice extracted from the text in Fig. 5.4.

At this stage, no relationship between frags and glyphs can be established, since a large number of glyphs are composed by two (or more) unconnected set of points. Moreover, it is allowed to draw one glyph over another, or two glyphs could simply be close enough to result connected after the digitalization process, if any. For these, and for a limited number of other reasons, we can only state that a frag may represent a glyph, a set of glyphs, or just a part of a glyph (see Fig 5.7). The “pop-arty” illustration (Fig 5.8) of the frags detected within a text concludes this section. Each frag within the text has been highlighted with a different random color.



*Figure 5.6:* An example of frag (courtesy of C.S. Bianchini). This is the first frag extracted from the slice in Fig. 5.5.



*Figure 5.7:* The complex relationship between frag and glyphs: a frag may represent a set of glyphs (left), a glyph (middle) or just a part of a glyph (right - the glyph is composed by three different frags).

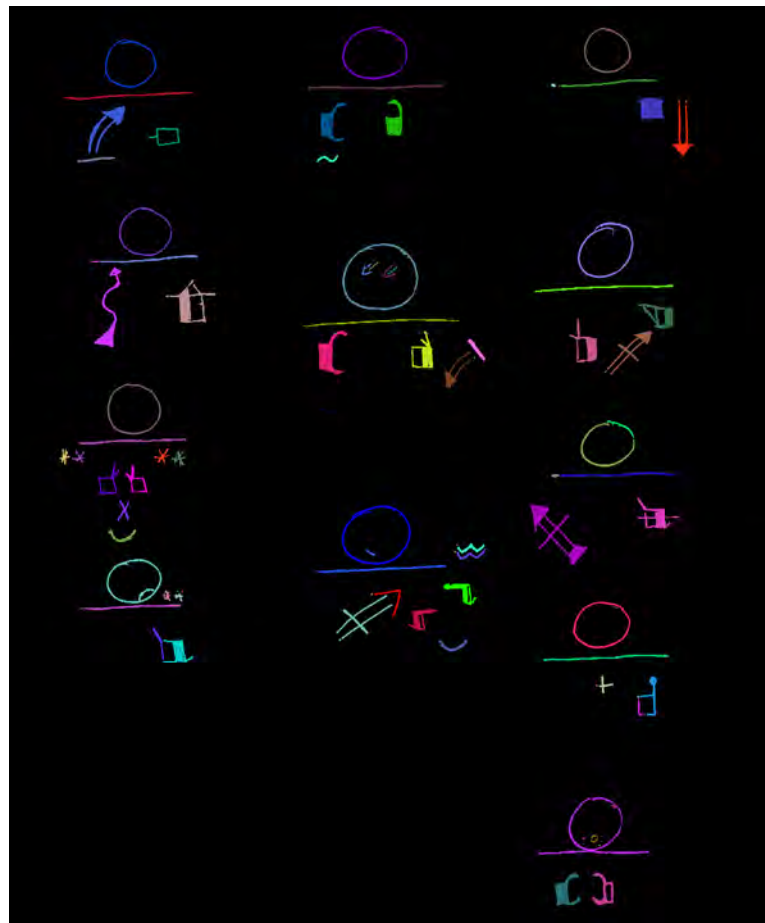


Figure 5.8: Set of frags detected within the text in Fig. 5.4. Each frag has been highlighted with a different random color.

### 5.3 OGR coding for SignWriting

The SW-OGR Engine mainly performs recognition by evaluating the geometric and topological features of the glyphs. Therefore, in order to carry out such evaluation, it was necessary to devise an alternative coding system for SignWriting, based on geometric and topological criteria only. Such coding is not intended to be used as an alternative to ISWA, but rather as a coding of the process (sequence of checks) needed to recognize glyphs. It is worth noticing that the code is independent from the software procedures which are actually used to perform the required steps. We further detail this issue. As illustrated in Chapter 2, the only available coding for SignWriting is the ISWA coding, which is based on linguistic and production criteria. However such criteria are semantic ones (e.g. we already know that a certain glyph is an hand) while we rather need “lexical” ones. Since we could not use the existing coding system, we devised a new one, namely the *OGR coding* for SignWriting.

Unlike ISWA identifiers, OGR identifiers encode the relevant geometric and topological features of the glyph they represent. A few examples of the features that the OGR coding takes into account are: the presence and the size of geometric shapes (circles, rectangles, etc..) detected within the image of the glyph, the presence of relevant convexity defects<sup>1</sup> within the convex hull and the presence of thick foreground clusters (see Fig. 5.9). Some features, like the width to height ratio of the glyph, are very easy to identify, while others may require more complex analyses. Using a wide range of image processing techniques, however, the SW-OGR Engine is able to automatically identify the OGR coding of most glyphs within a handwritten SignWriting text. This is the feature of the OGR coding system that actually makes the recognition possible, and it is indeed its main advantage.

Devising a OGR coding for a given set of glyphs is a very delicate task. Each

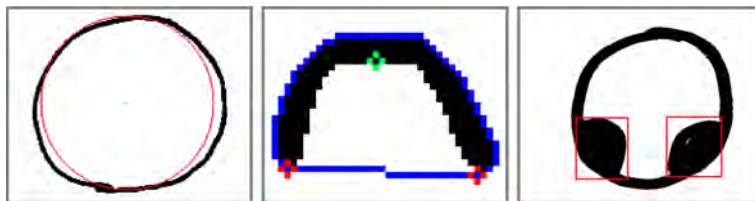


Figure 5.9: A few examples of the features encoded by an OGR identifier: from left to right: presence of geometric shapes, presence of convexity defects within the convex hull, presence of foreground clusters.

digit in the OGR code represents a feature that the SW-OGR Engine must evaluate in order to identify the glyphs within the set. Multiple requirements should be satisfied in order to produce a working OGR coding.

<sup>1</sup>In mathematics, the convex hull of a set  $X$  of points in the Euclidean plane is the smallest convex set that contains  $X$ . Any deviation of the set  $X$  of points from the convex hull can be considered as convexity defect.

- *Uniqueness*: One OGR code (i.e. a sequence of checks over the glyph) must identify one single glyph, otherwise the coding is ambiguous.
- *Robustness*: The OGR coding shall take into account the most recurring handwriting inaccuracies for each glyph and address them as much as possible (see Fig. 5.10).
- *Efficiency*: The number of checks necessary to build an OGR code should be kept to a minimum in order to simplify and speed up the recognition.

Failure to comply with the above requirements may have an impact of varying severity on the recognition. If the coding system is not unique, the recognition would not be possible at all. In fact, after having performed all the necessary checks over the image of a glyph, the SW-OGR Engine would be forced to choose between two or more glyphs with the same OGR code, without having further information to exploit. It is very important to notice that the uniqueness requirement only states that one OGR code must identify one single glyph, but the contrary is not necessary, and in some sense not even desirable. Actually, in some cases, it is very useful to have a glyph identified by multiple OGR codes. One of the most common reasons for this is to handle recurring handwriting inaccuracies by the users, which can dramatically alter the features of a glyph (see Fig. 5.10). It is worth noticing that this also addresses the robustness requirement.

At a significant extent, in any case, less severe consequences occur if the OGR coding is not robust. In fact, if (at least) the most recurring handwriting inaccuracies are not taken into account, the system would simply require a very high level of precision from the users, but it could theoretically be still working. On the other hand, a non-robust coding system, however, would force the users to handwrite symbols so that they are very similar to the digital glyphs, and this could prove a very tiring and frustrating job. Finally, failure to comply with the efficiency requirement would result in a coding which demands more checks over the glyph than necessary. A direct consequence of such inefficiency is a redundant, resource-wasting SW-OGR Engine. Inefficiency does not necessarily compromise the overall functioning and recognition accuracy of the application. However, one of the purposes of SW-OGR is to perform real-time glyph recognition, and speed is a key requirement for the application, thus inefficiency cannot be accepted.

The shapes of the glyphs may be very different from category to category, and even from group to group. Therefore, the features encoded within the OGR identifiers vary accordingly. For example, the presence of one or more triangles is valuable information for glyphs related to movements, but it is completely useless when encoding the features of a hand configuration. In order to avoid wasting time and resources on useless checks, the OGR coding could benefit of the knowledge of the ISWA category or group it is referring to. In other words, the SW-OGR Engine should not look for (and encode) fingers, if the glyph being recognized is a facial expression.

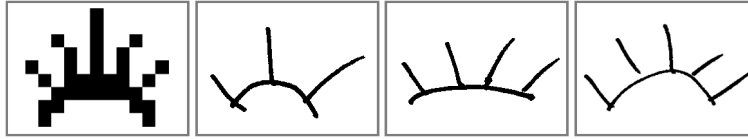


Figure 5.10: Glyph representing the shape of an eye (ISWA-2010: 04-02-006-01-01-01). The original digital version is visible on the left, along with some examples of drawing inaccuracies, found within our datasets.

Therefore, the very first information that the OGR coding should carry is about the *OGR area* (or simply *area*) of the glyph.

*An OGR area is a category or group within the ISWA whose glyphs present similar geometric features.*

The definition of area is intentionally vague (“a category or group”)<sup>2</sup> because in some cases a category may contain glyphs which present very similar features (e.g. Category 1 - Hands), thus they can all be placed into the same area. In other cases, however, the same category may contain glyphs which are very different, according to the group they belong to (e.g. Category 4 - Head & Faces). In such cases, an area corresponds to a group.

Categories and groups are concepts that have been introduced within Sutton’s ISWA, and, as explained in Chapter 2, their definition is based on linguistic and production (semantic) criteria. Such criteria are meaningless within the OGR coding context. As a consequence, in order to pick out the area of a glyph, it is necessary to isolate its geometric and topological features and use them to identify its category and/or group. For example, to check if a glyph belongs to the Contacts area, it is important to check for the existence of particular shapes which are typical of the glyphs belonging to Group 11 - Contacts, in Category 2 - Movement. Such shapes are asterisk symbols, plus symbols, etc. Moreover, it is necessary to check whether the size (e.g. width-to-height ratio) of the detected shapes respects a number of constraints or not.

Tab. 5.1 reports the full list of areas defined within the OGR coding system. For each area, the table reports:

- The code, i.e. a concise string, used to refer to the area during the design and development phase.
- The ISWA category or group associated to the area.
- The list of checks necessary to assess if a given glyph belong to the area. The check are performed over the image of the glyph, and they include:

<sup>2</sup>Categories and groups are the fundamental sets (and subsets) of glyphs in SignWriting. They have been introduced in Chapter 2.

- *Shape*: detection of shapes within the image.
- *Shape cardinality*: the number of detected shapes within the image.
- *Size*: different analyses of the size of the image (width-to-height ratio, minimum or maximum width or height, etc.).
- *Position*: position of the image with respect to a reference point or image.
- *Containment*: containment of the image with respect to a reference shape or image.

#### Areas defined within the OGR coding system

Code	Description	Checks
HaC	Hand Configurations	Shape (CIRCLE, RECTANGLE, ...). Size.
HaM	Hand Movements	Shape (TRIANGLE). Size.
ArP	Arm Positions	Shape (LINE). Size.
ArM	Arm Movements	Shape (TRIANGLE). Shape cardinality. Size.
ShP	Shoulders Positions	Shape (RECTANGLE, ...). Size.
ShM	Shoulder Movements	Shape (CIRCLE, RECTANGLE, ...). Size.
HeE_HE	Head Circles Head Contacts	Shape (CIRCLE). Size.
HeE_EY	Facial Expressions: Eyes	Containment within head. Position.
HeE_NO	Facial Expressions: Nose	Containment within head. Position.
HeE_MO	Facial Expressions: Mouth	Containment within head. Position.

*Continued on next page*

**Areas defined within the OGR coding system** – *Continued from previous page*

Code	Description	Checks
HeM	Head Movements	Shape (CIRCLE, TRIANGLE) Size. Position.
Co	Hand Contacts	Shape (ASTERISK, PLUS, ...). Size.
Dy	Dynamics	Shape (ARC, TILDE). Size.
Cr	Coordination	Shape (ARC). Size.
Pu	Punctuation	Shape (RECTANGLE). Size.

*Table 5.1:* Areas defined within the OGR coding system. For each area, the table reports the code, the ISWA category or group associated to the area, and the checks that SW-OGR performs, in order to check if a glyph belongs to the area.

Each area contains glyphs characterized by very different geometric features. For such reason, each area has a different type of OGR coding, according to the features that it is worth detecting within the images of the glyphs. The different coding may vary from simple to complex, according to the cardinality of the glyphs within the areas. In fact, an area which contains a very high number of similar symbols, requires a very high number of checks to univocally identify each of them. For example, the OGR coding of a very small area, such as Facial Expressions - Nose, features only two digits. A very dense area, on the other hand, such as Hand Configurations, has a variable-length OGR code that may extend up to 145 digits, if the encoded hand configuration has a high number of details (i.e. fingers, intersections between fingers, etc.).

The following sections (Section 5.3.1 and Section 5.3.2) show two representative OGR areas, illustrating their OGR codings and explaining the procedure to build them.

### 5.3.1 Hand Contacts (Co)

The present section shows the OGR coding for the Hand Contacts area, and explains the procedure we followed to build it. Such area contains an



average number of glyphs (85 symbols) and it is a representative example of our efforts to build a OGR coding system covering each SignWriting glyph. The first step to devise an OGR coding for a given area is to identify the set of relevant image features to be extracted from the glyphs belonging to the area. As a consequence, it is useful to gather such glyphs together to visually compare them and identify both common elements (to be checked first in order to first identify the area) and peculiar ones (to distinguish glyphs within the same area). Fig. 5.11 shows the whole set of glyphs within the Hand Contacts area.

For the convenience of the reader, it is important to shed some light on the production meaning of the contact symbols. The choice of the symbol indicates a different interaction between a hand and another part of the body. An asterisk symbol is used for simply touching the contact region, a circle with a dot inside for brushing along the region and then leaving it, a spiral for rubbing the region and not leaving, a hash symbol for striking the region, and finally a plus sign for grasping the region (usually the other hand). From the previous list, it is possible to notice the high level of production detail featured by SignWriting.

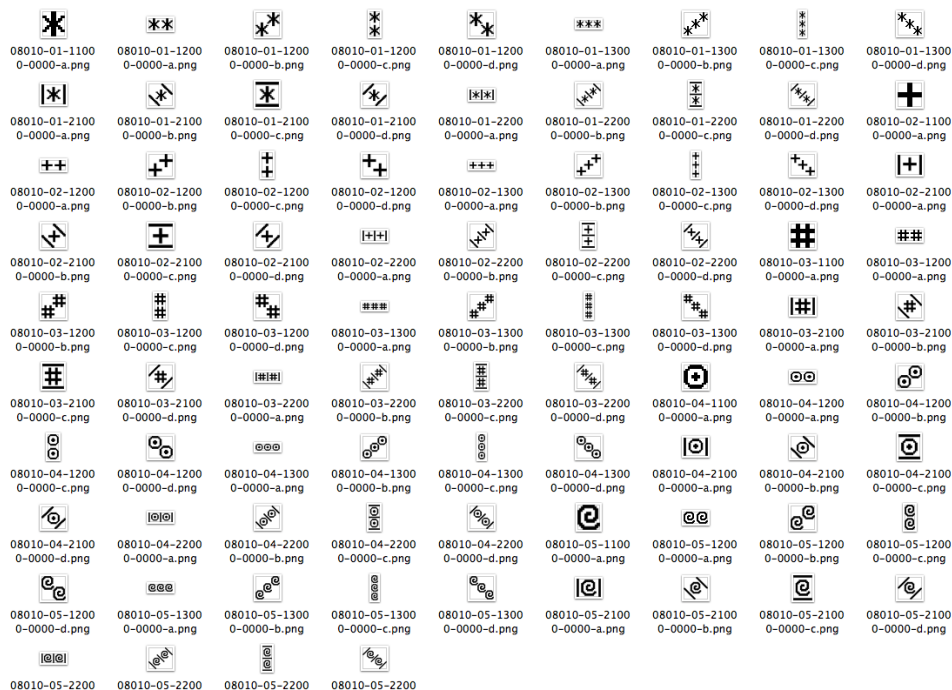


Figure 5.11: The whole set of glyphs within the Hand Contacts area (85 symbols).

Even if the total number of glyphs is not that high (85 symbols), it is very inefficient to work on the whole set. In fact, in most cases, there are features that are very easy to detect, but they are responsible for a dramatic increase of the cardinality of the glyphs within a given area. In the case of Hand Con-

tacts, we preferred to ignore such features, and to detect them at the end of our recognition procedure. One of such features is the rotation. In our case, ignoring the rotation feature, the symbols to be identified dropped from 85 to 25 (decreasing by 70.5%). The set of remaining symbols is visible in Fig. 5.12.

From the point of view of the implementation of SW-OGR, ignoring a feature (such as the rotation), implies designing recognition procedures which are invariant with respect to that feature. In the case of Hand Contacts (and in many other OGR areas), we implemented recognition procedures which are rotation-invariant. This goal was achieved by avoiding to use information about the exact location of the features within the image. We rather used the detection of the presence (and the evaluation of the characteristics) of some relevant cues over the image of the glyph, and we quantitatively analyzed the results. As a general example, we used techniques such as the generalized Hough transform for the detection of curves (Duda & Hart, 1972) to hypothesize the presence of a head, and the classification of the points of interests (end, branch and cross points) of a thinned image proposed by (Sarfranz, 2005) to identify specific features of hand configurations and movements.

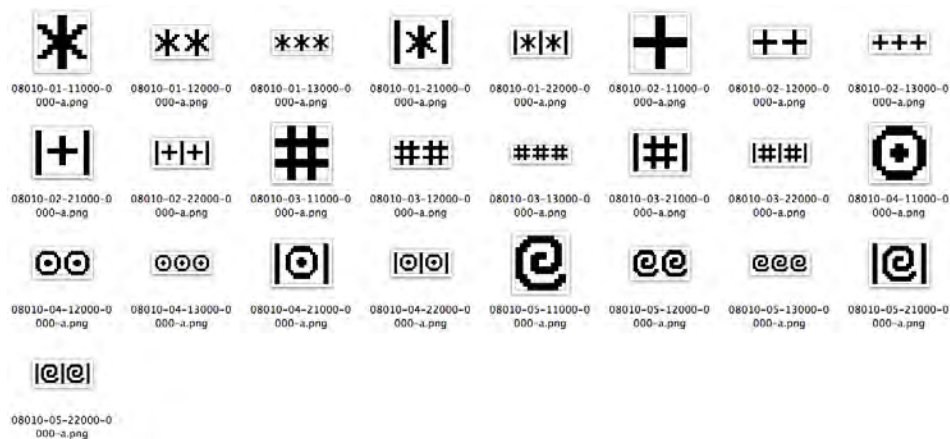


Figure 5.12: Glyphs within the Hand Contacts area, without rotations (25 symbols).

Analyzing the remaining set of glyphs, we searched for the most outstanding feature to operate an initial division. We observed that, separating the glyph according to the *base shape* detected within their images, we could split our set into 5 subsets of equal cardinality. We assigned a code to each base shape, as described in Table 5.2. Such code is the first digit of the OGR coding of the Hand Contacts area.

#### Hand Contacts area - *base shape* check details

Shape	Code	Image
PLUS	1	+
HASH	2	#
ASTERISK	3	*
SPIRAL	4	@
CIRCLE	5	⊙

Table 5.2: Hand Contacts area - *base shape* check details. For each shape, the table reports the name, the OGR code and an example image.

Visualizing the organization of the area, shaped by the different geometric checks, is very helpful both for computer scientists working for the OGR coding, and for the readers of the present work. In order to do so, we used a tree-like diagram, namely a *OGR-tree*. Fig. 5.13 shows the organization of a typical OGR tree. The root of the tree carries the name of the area, and its code, e.g. *Hand Contacts*. It represents the starting point for the following evaluations. Each level of the tree represents a different feature of the image (e.g. *base shape*), the more a level is close to the root, the earlier the associated feature is evaluated. Internal nodes on the same level represent different results for the same evaluation (e.g. *Base Shape: PLUS*, *Base Shape: HASH*, etc.). Each internal node within the diagram carries information about the associated evaluation, the associated result, and about the code (one or more digits) associated to the result. The leaf nodes of the tree represent the glyphs within the area. Each leaf carries the image of the glyph, as well as its final OGR code, and its ISWA-BIANCHINI code.



Figure 5.13: Organization of a OGR tree.

The OGR tree in Fig. 5.14 shows the state of the Hand Contacts area after

the evaluation of the first feature. As anticipated, the glyphs have been divided according to the base shape. Looking at the OGR codes written on the leaf nodes, it is evident that such division is not sufficient to assign a unique OGR identifier to each glyph, since many of them have the same OGR identifier. Therefore, it is necessary to detect more features.

A further level of specification can be added by counting the *repetitions* of



Figure 5.14: Hand Contacts area after the *base shape* check.

the detected base shape, with the possible values being 1, 2 or 3 repetitions. The detection of such feature divides each of the 5 subsets resulting from the first check into 3 further subsets. Fig. 5.15 shows the state of the Hand Contacts area after the evaluation of the second feature. A small number of glyphs, (i.e. [OGR: 1.3], [OGR: 2.3], etc..) have already been identified in an univocal way by the repetition check. Most glyphs, however, still present ambiguity, and need further specification. Looking at Fig. 5.15, please notice that some internal nodes and leaf nodes are hidden for the sake of space and readability. Please notice also that the OGR codes on the leaf nodes have been updated, including the results from the repetition check.

The remaining ambiguous glyphs are almost equal, but some of them present one or more lines running along the sides of the base shape. This is actually the only feature that can be exploited to remove the remaining ambiguity. It is evident, from Fig. 5.16, that after the line check, each glyph is identified by a different OGR code, thus fulfilling the uniqueness requirement of the OGR coding system. Again, some internal nodes and leaf nodes are hidden from Fig. 5.16, and the OGR codes on the leaf nodes have been updated.

It is worth reminding that almost each leaf node in Fig. 5.16, has 4 possible rotations. The rotations have been hidden from our initial set of glyphs to avoid useless overhead, but such feature must be encoded within the OGR code. Therefore the rotation, with its 4 possible values, is the last digit of the OGR coding system for the Hand Contacts area.



Figure 5.15: Hand Contacts area after the *repetition* check. A number of internal nodes and leaf nodes are hidden for the sake of space.



Figure 5.16: Hand Contacts area after the *line* check. A number of internal nodes and leaf nodes are hidden for the sake of space.

The following list summarizes the sequence of features exploited to build the OGR coding for the Hand Contacts area.

1. Base shape (PLUS, HASH, ASTERISK, SPIRAL, CIRCLE).
2. Repetition (1, 2, 3).
3. Line (NO, YES).
4. Rotation ( $0^\circ$ ,  $45^\circ$ ,  $90^\circ$ ,  $135^\circ$ ).

### 5.3.2 Hand Configurations (HaC)

The OGR coding of the glyphs belonging to Hand Contacts area is rather simple (4 digits), since the area contains 85 symbols. The OGR coding shown in the present section is far more complex (24 to 145 digits), since it encodes the features of the glyphs within the Hand Configurations area, which contains more than 23.000 symbols. Such area boasts the highest number of glyphs in the OGR coding system. In fact, it proved one of the toughest challenges we faced during our work.

The glyphs of the Hand Configurations area present features that are very easy to detect, such as the color fill (white, black & white, black), as well as other, more fine-grained features, such as those related to the fingers. Such features are fairly difficult to draw, so the image produced by the user can be very inaccurate. Moreover (and also due to this) such features are rather hard to detect, thus they require a very thorough analysis of the image. Therefore, reaching the *aurea mediocritas* between accuracy, efficiency and robustness is crucial to build a working OGR coding for the Hand Configurations area.

During the initial stage of our work on this area, we followed the same approach illustrated in Section 5.3.1. First of all, we ignored the rotation feature<sup>3</sup>, and we postponed its detection to the final steps of the recognition. Most symbols in this area have 8 possible rotations ( $0^\circ$ ,  $45^\circ$ ,  $90^\circ$ ,  $135^\circ$ ,  $180^\circ$ ,  $225^\circ$ ,  $270^\circ$ ,  $315^\circ$ ), so the working set of symbols dropped from 23.136 to 2.892 units (decreasing by 87.5%).

Since the set of symbols is still very large, illustrating our work by showing detailed views of the OGR tree might not be the best option. For this reason we will simply enumerate, discuss and illustrate the features we decided to evaluate within the glyphs, in order to build our OGR coding for this area.

As for the Hand Contacts area, the first feature we decided to isolate is the *base shape* detected within the glyphs. Such base shapes are used to represent the palm of the hand, and they divide the working sets of glyphs into subsets of varying cardinality. Tab 5.3 show the possible base shapes, and

---

<sup>3</sup>Similarly to what stated for Hand Contacts (Section 5.3.2), ignoring a feature (such as the rotation), implies designing recognition procedures which are invariant with respect to that feature.

the associated code. This is the first digit of any OGR code within the Hand Configurations area.

#### Hand Configurations area - *base shape* check details

Name	Code	Shape	Image
SQUARE	1	□	
RECTANGLE	2	▭	
CIRCLE	3	○	
HOUSE	4	▧	
R-HOUSE	5	▨	
TRAPEZIUM	6	▭	
SPIRAL	7	⌀	

*Table 5.3:* Hand Configurations area - *base shape* check details. For each shape, the table reports the name, the OGR code, an image of the shape, and a number of example glyphs.

The second feature we identified is the color filling of the base shape. Such color filling identifies the *orientation* of the palm of the hand, and it may assume three possible values: *WHITE*, *BLACK&WHITE*, *BLACK*. Table 5.4 shows more details about the orientation feature. The evaluation of such feature concludes our assessments related to the base shape of the glyph. The following evaluations are all focused on the fingers. Notice that we prefer to use the word *appendices* to refer to the fingers, since such word has no meaning in the OGR context.

#### Hand Configurations area - *orientation* check details




Shape	Code	Image
WHITE	1	
BLACK&WHITE	2	
BLACK	3	

Table 5.4: Hand Configurations area - *orientation* check details. For each orientation, the table reports the name, the OGR code and a number of example images.

The first evaluation performed over the appendices identifies their number (*appnum*). Each glyph may have from 0 up to 5 appendices, and such number is the third digit within the OGR code.

The following evaluation aims at identifying whether the appendices (if any) are detached or attached to the base shape. We named it the *attachment* check. Since it is not possible to have both attached and detached appendices on the same symbol, the check may only return two possible values: *ATTACHED* and *DETACHED*. More details are provided in Tab. 5.5.

#### Hand Configurations area - *attachment* check details


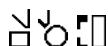
Shape	Code	Image
ATTACHED	1	
DETACHED	2	

Table 5.5: Hand Configurations area - *attachment* check details. For each case, the table reports the name, the OGR code and a number of example images.

The following evaluation is about the location of the appendices (*apploc*). We define an *outer* appendix as an appendix which extends on the space outside of its base shape, otherwise it is an *inner* appendix. Most glyphs have outer appendices, but there are also less frequent cases, such as inner and mixed (both inner and outer) appendices. Tab 5.6 provides more details about the appendix location feature.

#### Hand Configurations area - *apploc* check details







Shape	Code	Image
NO	0	
INNER ONLY	1	
OUTER ONLY	2	
MIXED	3	

Table 5.6: Hand Configurations area - *aploc* check details. For each case, the table reports the name, the OGR code and an example image.

The fingers of the human hand may assume different shapes: they can be extended, bent, etc. As a result, the appendices of the glyphs in the Hand Configuration area can feature different shapes. The appendices can be *STRAIGHT* or *DOT* when they represent an extended finger, and *BENT* or *STRAIGHTDOT* when they represent a bent finger. Most glyphs feature *STRAIGHT* or *BENT* shapes, while the other shapes are rather uncommon. Tab 5.7 shows the possible shapes of the appendices, showing a number of examples for each of them.

#### Appendix shapes within the Hand Configurations area





Shape	Description	Image
STRAIGHT	The appendix is a straight line.	
STRAIGHTDOT	The appendix is a straight line, terminating in a dot.	
DOT	The appendix is a dot or a square.	
BENT	The appendix is a bent line or a curved line.	

Table 5.7: Appendix shapes within the Hand Configurations area. For each shape, the table reports the name, a short description and a number of example images.

The shapes of the appendices are the next feature to be evaluated within our OGR code, according to the above classification. In fact, we decided to count the number of appendices dividing them according to their shape. More specifically, the appendix shape (*appshape*) check produces a 4-digit string, which is built in the following way:

- 1st digit: the total count of STRAIGHT appendices.
- 2nd digit: the total count of STRAIGHTDOT appendices.
- 3rd digit: the total count of DOT appendices.
- 4th digit: the total count of BENT appendices.

As an example, if the result of the *appshape* evaluation is *2001*, the analyzed glyph features two STRAIGHT appendices, and one BENT appendix. More examples are available in Fig. 5.17, please notice that there are three different symbols which are encoded by the same string, i.e. *3000*. This is a very frequent case in the Hand Configurations area.

It is important to observe that any evaluation performed so far takes into



Figure 5.17: Results of the *appshape* check, applied to different glyphs.

account the features of the glyph as a whole. The OGR coding devised so far encodes information about the image at a very high level (e.g. the color filling, the number of appendices, etc.). However, since the symbols in this area are very similar, the OGR coding we just illustrated is very far from being unique. Before going further, it is worth recapitulating the features evaluated until this point.

1. Base shape (SQUARE, RECTANGLE, CIRCLE, ...).
2. Orientation (WHITE, BLACK&WHITE, BLACK).
3. Appnum (0, 1, 2, 3, 4, 5).
4. Attachment (ATTACHED, DETACHED).
5. Apploc (NO, INNER, OUTER, MIXED).
6. Appshape (4-digit string).

In few, rare cases, such OGR code is enough to univocally identify a single glyph: it is the case of those symbols which feature no appendix at all, and a few other exceptions. In all the remaining cases, however, this code simply

identifies subsets of symbols whose cardinality is in the order of tens. Therefore, analyzing the glyph side by side, appendix by appendix, is necessary to identify it in a univocal way. We need answers to questions like:

- How many appendices start from a given side?
- What kind of angle does a given appendix form with its starting side?
- Does a given appendix intersect any other appendices?

A very accurate analysis is necessary, and, as a consequence, a very detailed OGR code. The more appendices are detected within the symbol, the longer the code shall be. This is the reason why the OGR code for the Hands Area has a *fixed part*, which is the one we illustrated so far, and a *variable part* which is the one we are about to illustrate, which is directly dependent on the number of appendices. The variable part of the code is very detailed, and it carries many benefits: first of all, it provides the required level of knowledge over the details of the appendices. Moreover, it is determinant to detect the rotation of the glyph, and a number of other important features. The variable part of the OGR identifier encodes, for each side of the base shape, the related appendices. The sides are encoded following a fixed order, i.e. clockwise order, starting from the top-left corner of the base shape (see Fig. 5.18). The content of each side (i.e. the coding of its appendices) is enclosed between square brackets. For example if the coding of a side is `[ ]`, then the side has no related appendices. If content is found between the brackets, this means that the sides has one or more appendices. Fig 5.18 illustrates the encoding of the sides for a glyph which features a SQUARE base shape and a single appendix. In Fig. 5.18, please notice the encoding start point and the encoding direction: they determinate the evaluation order of the sides, and of the appendices they contain. The evaluation order is shown as a numeric counter near the sides (Side 01, Side 02, etc.).

For each side, appendices are evaluated following the encoding direction, and the coding of each appendix is enclosed within round brackets. For example, the variable part of the OGR coding of the glyph in Fig. 5.18 has the following form:

```
[ ] [ ] [(Appendix coding)] [(Appendix coding)]
```

The first two sides have no appendices, while the last sides share a common appendix. From the above code it is evident that when an appendix starts from a corner between two sides, it is encoded on both sides. We made such choice since we observed that, when an appendix starts from a corner between two sides, it is necessary to evaluate its relation with both of them, otherwise the coding may present ambiguities.

Each appendix is encoded by a 6-digit string (enclosed by round brackets). Such string encodes any relevant features of the appendix. As a convention,

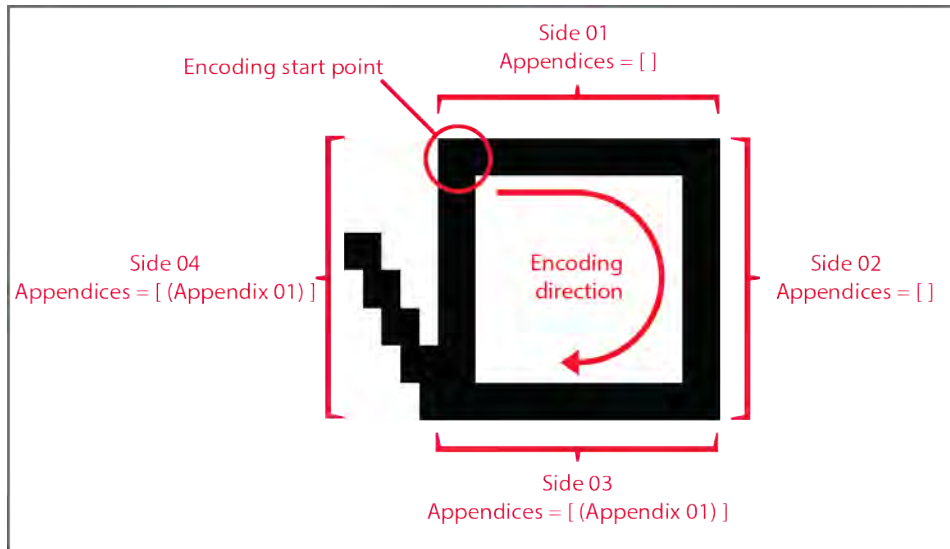


Figure 5.18: Side encoding for a glyph which features a *SQUARE* base shape.

we identified the *start point* as the point of the appendix which is closer to the side, and the *end point* as the farthest one. The encoded features are the following:

- *oncorner*: this digit encodes whether the start point of the appendix is on (or near) a corner of the base shape (see Fig 5.19). Possible values:
  - 1 if the appendix is on a corner
  - 0 otherwise
- *inprojection*: this digit encodes whether most of the length of the appendix is contained between the projection of its related side, Fig 5.20 explains this evaluation better than a thousand words. Possible values:
  - 1 the appendix is within the projection
  - 0 otherwise
- *location*: this digit encodes whether the appendix lies within the base shape. Possible values:
  - 1 the appendix is internal
  - 2 the appendix is external
- *shape*: this digit encodes the shape of the appendix, according to the codes identified in Tab. 5.7.
- *stangle*: this digit encodes the angle between the appendix and its related side. Possible values:
  - 1 if the angle is acute ( $0^\circ$  to  $80^\circ$ )

- 2 if the angle is straight ( $81^\circ$  to  $99^\circ$ )
- 3 if the angle is obtuse ( $100$  to  $180^\circ$ )
- 0 if the angle is wider than  $180^\circ$
- *contact*: this digit encodes the contacts between appendices. Possible values:
  - 0 if the appendix has no contacts
  - 1 if the appendix shares its start point with another appendix
  - 2 if the appendix shares its end point with another appendix
  - 3 if the appendix intersects another appendix
  - 4 if the appendix has two or more contacts

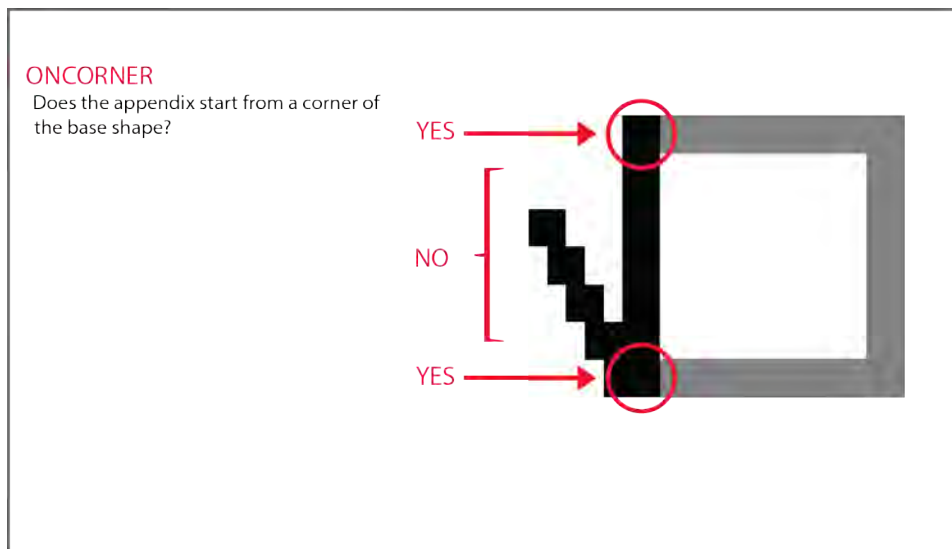


Figure 5.19: Appendix encoding: *oncorner* evaluation.

Fig. 5.22 illustrates a summary of the evaluations performed on the appendices. Concluding the example of the glyph in Fig 5.18, we report its full OGR code (both fixed and variable part).

```
1.1.1.1.2.1000 [ ] [ ] [(1.0.2.1.0.0)] [(1.1.2.1.3.0)]
```

The above identifier is a rather simple one, the code might get far longer and complicated than this, as the number of appendices within the symbol increases. Despite the coding may seem excessively detailed, removing even one of the encoded features would cause ambiguity within the coding. Evaluating the features illustrated in the present section we were able to build a working, non-ambiguous OGR coding for the glyphs within the Hands Configurations area.

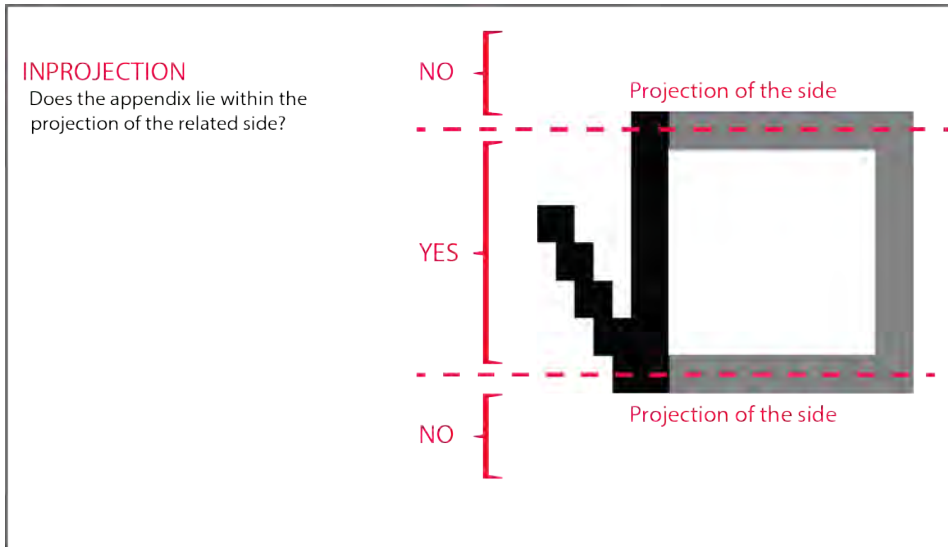


Figure 5.20: Appendix encoding: *inprojection* evaluation.

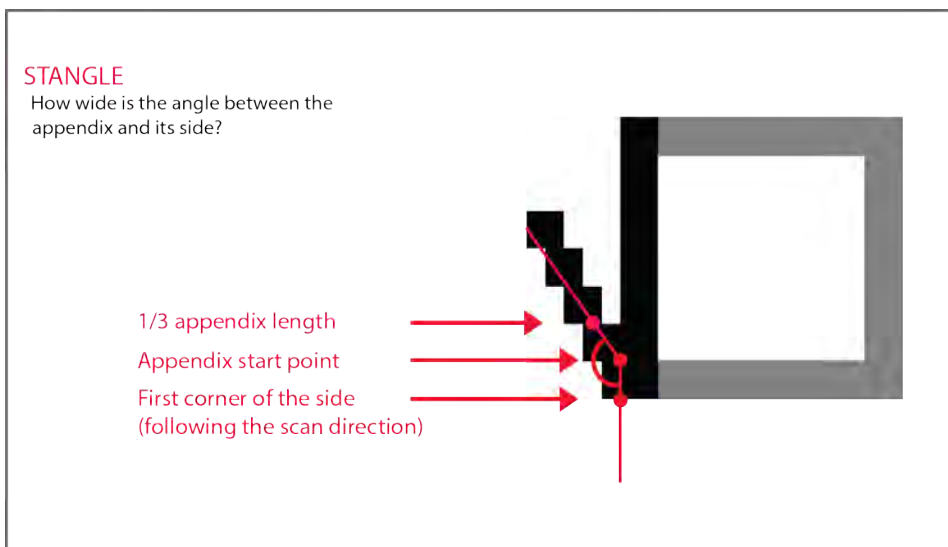


Figure 5.21: Appendix encoding: *stangle* evaluation.

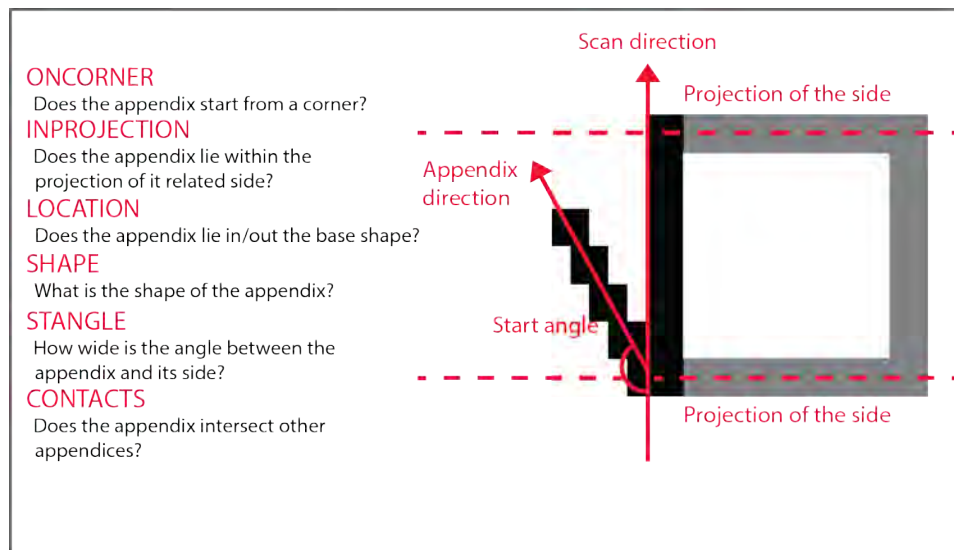


Figure 5.22: Appendix encoding: summary of all evaluations.

As mentioned earlier, a high level of detail within the coding can result in a general vulnerability to drawing inaccuracies. The *oncorner* check on the appendices, for instance, is one of the most critical evaluations, since there is a very high number of glyphs which feature appendices that should be drawn near corners (see Fig 5.23), but not exactly on them. Identifying the same glyph using two or more OGR codes quickly solves issues like this one.



Figure 5.23: A set of glyphs belonging to the Hand Configurations area. Such symbols can be subject to drawing inaccuracy: users may draw the bottom-right appendix of each glyph either on the corner or on the side.

## 5.4 Interoperability between coding systems

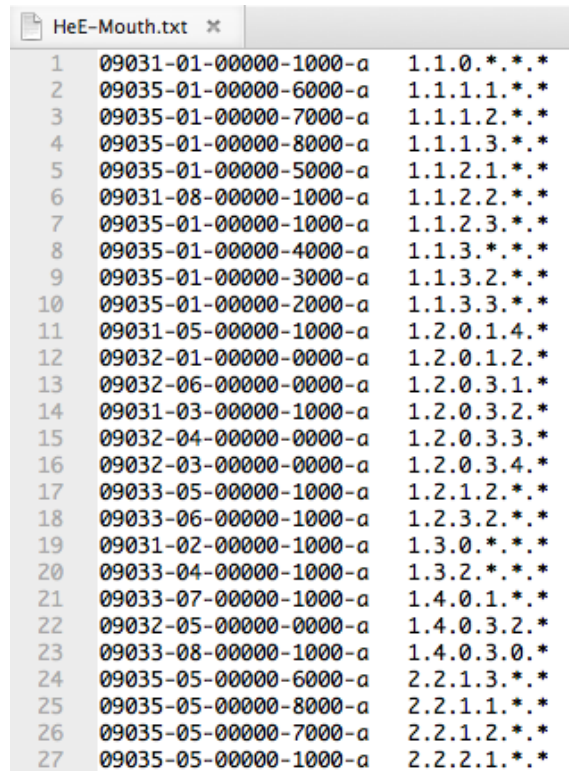
The OGR coding system shown in the previous sections is in all respects an alternative coding system for SignWriting. It can be used to encode any symbol within a SignWriting text. However, this is not its purpose. We created it as a SW-OGR Engine internal coding. As mentioned before, the main advantage is that OGR codes can be automatically identified by the SW-OGR Engine, simply analyzing the image of a glyph. However, once the glyph is recognized, it cannot be stored using the OGR code. First of all, it would be highly inefficient: think, for example, about a 145-digit

code for a glyph belonging to the Hands Area. Moreover, any research team working with SignWriting, and any digital artifact which is somehow related to SignWriting uses the ISWA coding. In summary, storing the recognized glyphs using their OGR coding would result in a general inefficiency, and in the total lack of interoperability with other teams or digital artifacts. This would dramatically limit the value of our work, and undermine the benefits of the SW-OGR Engine.

For the above reasons, all of our OGR trees carry a double information for each glyph, namely its OGR code(s), and the ISWA code. Processing such trees with simple regular expressions, we generated a number of mapping tables (one for each area), which convert OGR codes to ISWA-BIANCHINI codes. The regular expressions are used to remove all but the leaf nodes of the OGR trees, since each one of them carries an association between one (or more) OGR code(s) and one ISWA code, so they are the only piece of information that matters for the purpose of building the mapping table. The regular expressions are also used to format the data within each leaf node in a machine friendly format, i.e. each row of the table carries an ISWA identifier and a OGR identifier, separated by a tabulation character.

Summarizing, mapping tables are simple text files, like the one shown in Fig 5.24. Each row of a mapping table contains a ISWA-BIANCHINI code (left side), followed by its associated OGR code (right side). Please notice the asterisks within the OGR codes, on the right side of Fig 5.24. Such symbols work as a wildcard, and they are used when the encoded glyph has already been uniquely identified by the first detected features, so the encoding of the last features is useless.





Line	ISWA-BIANCHINI Code	OGR Code
1	09031-01-00000-1000-a	1.1.0.*.*
2	09035-01-00000-6000-a	1.1.1.1.*.*
3	09035-01-00000-7000-a	1.1.1.2.*.*
4	09035-01-00000-8000-a	1.1.1.3.*.*
5	09035-01-00000-5000-a	1.1.2.1.*.*
6	09031-08-00000-1000-a	1.1.2.2.*.*
7	09035-01-00000-1000-a	1.1.2.3.*.*
8	09035-01-00000-4000-a	1.1.3.*.*
9	09035-01-00000-3000-a	1.1.3.2.*.*
10	09035-01-00000-2000-a	1.1.3.3.*.*
11	09031-05-00000-1000-a	1.2.0.1.4.*
12	09032-01-00000-0000-a	1.2.0.1.2.*
13	09032-06-00000-0000-a	1.2.0.3.1.*
14	09031-03-00000-1000-a	1.2.0.3.2.*
15	09032-04-00000-0000-a	1.2.0.3.3.*
16	09032-03-00000-0000-a	1.2.0.3.4.*
17	09033-05-00000-1000-a	1.2.1.2.*.*
18	09033-06-00000-1000-a	1.2.3.2.*.*
19	09031-02-00000-1000-a	1.3.0.*.*
20	09033-04-00000-1000-a	1.3.2.*.*
21	09033-07-00000-1000-a	1.4.0.1.*.*
22	09032-05-00000-0000-a	1.4.0.3.2.*
23	09033-08-00000-1000-a	1.4.0.3.0.*
24	09035-05-00000-6000-a	2.2.1.3.*.*
25	09035-05-00000-8000-a	2.2.1.1.*.*
26	09035-05-00000-7000-a	2.2.1.2.*.*
27	09035-05-00000-1000-a	2.2.2.1.*.*

*Figure 5.24:* Detail of the OGR to ISWA-BIANCHINI mapping table for the Facial Expressions: Mouth area. Each ISWA-BIANCHINI code (on the left) is mapped to an OGR code (on the right).

Using such mapping tables, the SW-OGR Engine is able to recognize and encode a handwritten SignWriting text in a format that can be readable by any research team or present digital artifact in the world.



## Chapter 6

# Implementation of SW-OGR

### 6.1 Architecture of SW-OGR

#### 6.1.1 Concept

Before starting any engineering effort for the realization of the SW-OGR Engine, we worked to define the required operations to perform the recognition, from the acquisition of the input image to the creation of the output data. As anticipated in Section 4, we decided to be compliant with the 6-steps procedure which is distinctive of most modern OCR applications (Ahmed & Al-Ohali, 2000; Parker, 2010). As a remainder, such phases are: acquisition, pre-processing, segmentation, feature extraction, classification and post-processing. As stated in Chapter 4, no machine-learning approach was viable during the feature extraction and classification step. Moreover, the lack of statistical analyses on SignWriting compositions, and the high level of freedom featured by the system, made it impossible to perform the post-processing step automatically. As a consequence, the post-processing step is not managed by the SW-OGR Engine, it is rather performed with the aid of a human actor (as detailed in Section 4.1).

The present section provides a high-level description of the recognition procedure implemented by the SW-OGR Engine. The procedure is illustrated by the activity diagram in Fig 6.1. The purpose of the diagram is to grant the reader a general knowledge of the recognition procedure, therefore it does not provide any detail whatsoever about its activities. Each activity will be covered in full detail in the following sections.

As shown in Fig 6.1, the recognition starts as an image containing handwritten SignWriting glyphs is passed from the Data Acquisition Module in the User Interface to the SW-OGR Engine (Section 4). While the details regarding the adopted image processing techniques will be given in Section 6.2.1, we anticipate here a summary of the processing steps.

The acquired image undergoes a *pre-processing* step, whose purpose is to remove as many imperfections as possible. Such imperfections might be due to different factors, including poor source document quality, poor scanning equipment, etc. The most common imperfections the SW-OGR Engine faces

during this step are Gaussian noise and salt and pepper noise. After having applied a number of filters to correct the imperfections, the SW-OGR transforms the acquired image into a binary image (*binarization*). The binarization is performed by thresholding the images in an adaptive way, as detailed in Section 6.2.2. Working with a binary image enables or facilitates a number of very useful operations over the image, including (but not limited to): contour detection, thinning, etc.

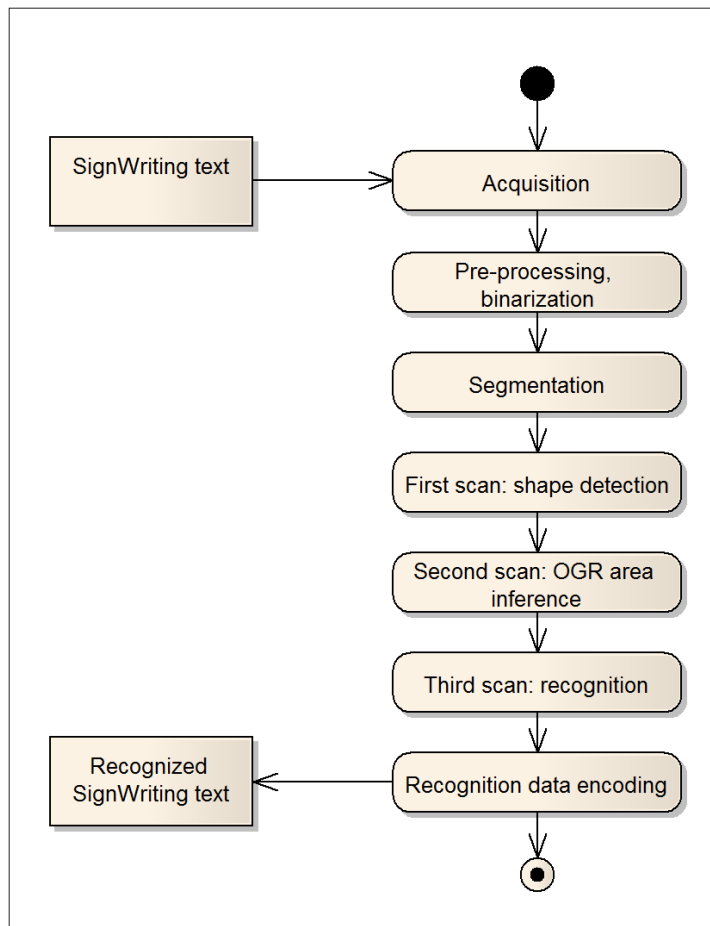


Figure 6.1: High-level activity diagram describing the recognition procedure implemented by the SW-OGR Engine.

After the preliminary operations, the image, i.e. the text, is divided in its component slices. Each slice is processed separately in order to identify its 8-connected components, namely the frags. Such operation is known as *image segmentation* in computer vision theory, and its purpose is to “simplify and/or change the representation of an image into something that is more meaningful and easier to analyze” (Stockman & Shapiro, 2001). In fact, any image operation illustrated from here on is performed slice by slice, frag by frag, following a *divide et impera* approach.

The *first scan* of the image aims at detecting any “interesting” shape within each frag. Such shapes may be circles, rectangles, asterisks, etc. The information gathered during this step is of primary importance during the following phases. In fact, the evaluation of the shapes allows the system to infer the area of each frag, and plays a key role during the ultimate recognition of the glyph.

During the *second scan*, each frag is analyzed, along with the information gathered during the previous step, in order to infer the OGR area with a reasonable degree of certainty. The evaluations performed during this step are conducted taking into account the uncertain relationship between frags and glyphs (see Section 5.2).

During the *third scan*, the actual recognition of the symbols takes place. A number of different advanced evaluations are performed on each symbol, and such evaluations are different according to the OGR area inferred during the previous step. At the end of this phase the SW-OGR Engine produces a set of recognized glyphs, encoded both in OGR and in ISWA-BIANCHINI format.

During the last step, the recognized glyphs are saved in multiple formats (typically an image format and/or an XML format), and returned back to the User Interface, for the human-assisted recognition review step.

### 6.1.2 Software architecture

Identifying the core recognition elements (Section 5.2) and the conceptual procedure (Section 6.1.1) allowed us to design the software architecture of the SW-OGR Engine. Before covering each step of the recognition procedure in full detail, it is important to provide the reader a good knowledge of such software architecture.

First of all, since the SW-OGR Engine contains a fair number of classes, it is appropriate to analyze its package diagram, which provides a global snapshot of the architecture of the system. The package diagram is reported in Fig 6.2. For each package, the diagram shows its dependencies with the other packages (inter-package dependencies), and a set of representative classes. Class details and class dependencies are out of the scope of such diagram, and will be covered later.

The `beans` package contains the implementation of the core recognition entities discussed in Section 5.2. In other words, the package contains the classes designed to store data about texts, slices, frags, and glyphs. This package also contains the `coding` class, whose purpose is to store data about the ISWA coding of a glyph. The classes within this package have very tight dependency relationships, therefore, no class could have a complete specification without the others.

The beans rely on the `util` package to perform a very large number of operations, ranging from image segmentation to the evaluations required to detect shapes within the frags. Actually, `util` is used by any package in the application, since it provides the necessary business logic to perform a

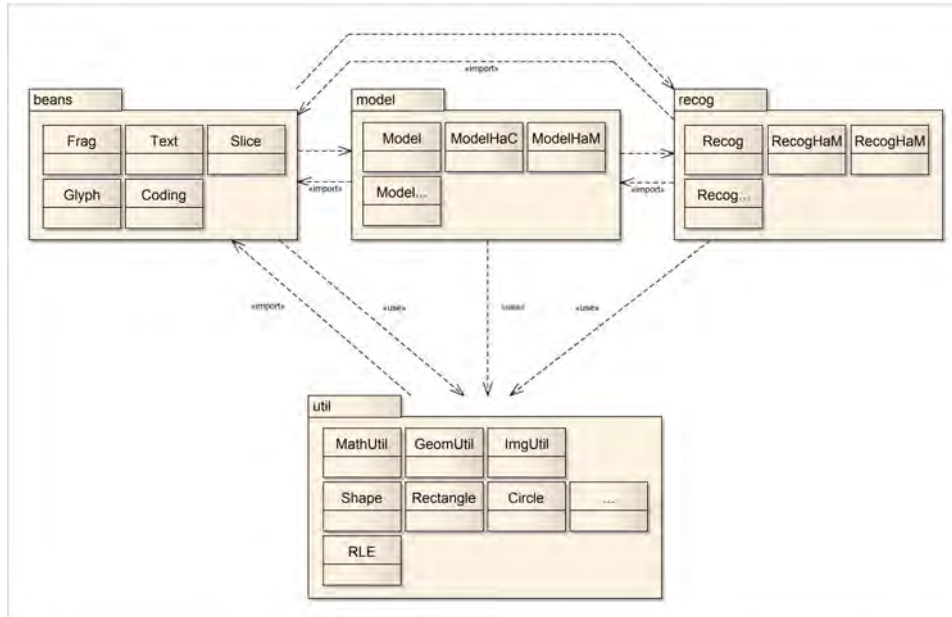


Figure 6.2: Package diagram of the SW-OGR Engine.

very large number of operations. Most of the `util` package is, in fact, a library of mathematical, geometric and image-oriented functions. This makes `util` one of the most useful (and definitely the most reusable) package in the whole application. The package also contains classes designed to carry different data, such as those for storing shapes (circles, rectangles, asterisks, etc), or those for storing the run-length encoding<sup>1</sup> (Lynch, 1985) of an image. A number of functions of the `util` package carry out operations directly on slices, frags and glyphs. This explains the import dependency towards the `beans` package in Fig. 6.2

The `beans` depend on the `model` package for any business logic dealing with the inference of the OGR area of each frag. Each class within the `model` package is designed to check the membership of a frag with respect to a given OGR area. For example, the `ModelHaC` class will assess whether a given frag may belong to the Hand Configurations area or not. The classes within the `model` package make large use of the `util` package to perform their evaluations.

Finally, the `beans` depend on the `recog` package for any business logic built to provide the actual recognition of the symbols. Each class within this package is designed to identify the OGR coding (and, as a result, the ISWA coding) of a glyph which has been found belonging to a given OGR area. For

<sup>1</sup>Run-length encoding is a very simple form of data compression in which *runs* of data (i.e. sequences in which the same data value occurs in many consecutive data elements) are stored as a single item containing the value and the count of occurrences. This is most useful on data containing consecutive occurrences of the same value. In our case, it is very effective, since the recognition procedure deals with binary images (which may present only two possible values for each pixel).

example, the `RecogHaC` class will identify the OGR coding of any glyph belonging to the Hand Configurations area. Such classes perform a large number of advanced image evaluations, making large use of the `util` package. Both the `model` and the `recog` package have no intra-package dependency, i.e. their classes are mutually independent.

### The beans package

The present section provides an in-depth view of the `beans` package, describing its classes and their relationships. The class diagram of the package is available in Fig. 6.3. For the sake of space and readability, the diagram does not show a number of operations (e.g. get and set operations) and fields which are of minor importance to understand the structure of the classes. As anticipated in Section 6.1.2, the classes within the `beans` package are the implementation of the core recognition entities; as a result, they present very tight mutual relationships, as shown in Fig. 6.3.

In order to describe the classes of the `beans`, it is worth starting from the `Text` class, since it is the highest-level class of the application, and since it controls the entire recognition process, from the acquisition of the image to the production of the output data. An instance of the `Text` class basically contains:

- The image of the text, and a field which keeps track of its visual organization (horizontal or vertical).
- The output image, i.e. the image containing the recognized glyphs. Such image, and the set of recognized glyphs is the actual result of the recognition.
- The list of the slices identified within the text. Such slices are instances of the `Slice` class: this explains the relation between the `Text` and the `Slice` class.

Among the most notable operations of this class, it is worth mentioning the ones for the pre-processing and thresholding of the text. Moreover, a number of operations are devoted to the detection and the management of the slices, which, according to the visual organization of the text, can be either rows or columns. Finally, a very important set of operations is composed by those for invoking the three image analysis steps which ultimately lead to the recognition of the text. Such operations are:

- `invokeFirstScan()`: invokes the first analysis on the image, whose purpose is to detect any interesting shape within the frags.
- `invokeSecondScan()`: invokes the second analysis on the image, whose purpose is to infer the OGR area of each frag.

- `invokeThirdScan()`: invokes the third analysis on the image, whose purpose is to actually recognize the symbols within the text.

Such operations are invoked within the `Text` class, but they are delegated to the slices, which in turn delegate them to their respective frags (see Fig. 6.3). Therefore, the actual evaluations are performed within the `Frag` class, but they are coordinated by the `Slice` class and by the `Text` class.



Figure 6.3: Class diagram of the beans package.

As mentioned before, any text is composed by a number of slices, the `Slice` class is designed to store any useful data about a single slice, including:

- A reference to the parent `Text` object, along with the boundaries (top left point and bottom right point) of the slice within the text.
- The image of the slice.
- Different structures to keep track and map any frag within the slice.
- Different structures to keep track and map any glyph within the slice.

Observing the fields of the `Slice` class, it is easy to understand why it is the one with most relationships within the package. Slices, in fact, are highly dependent on the text they are in, and on the frags (and later the glyphs)



that they contain. Most of the operations made available by this class are delegated by the `Text` class, such as those for the invocation of the three recognition scans (`invokeFirstScan()`, etc.).

Actually, there's more than the classic *divide et impera* rationale behind our delegation-based approach. In fact, performing operations on smaller elements, such as slices, or frags, instead of texts, grants a remarkable advantage: efficiency. In fact, as an example, there is no reason to perform operations such as segmentation on areas of the image which belong to no slice, since they are certainly empty (i.e. absence of relevant foreground).

The `Frag` class is designed to store any information about a given 8-connected foreground component detected within a slice. Such information are the following:

- A reference to the parent `Slice` object, along with the boundaries (top left point and bottom right point) of the frag within the slice.
- The image of the frag, and the set of its foreground points.
- Shapes (circles, rectangles, asterisks, etc.) and holes detected within the frag. The value of these fields is typically defined during the first recognition scan.
- A reference to the instance of the `Glyph` class associated to the `Frag`. Such association is defined during the second recognition scan, as the SW-OGR Engine infers the OGR area of the frag.

A relevant part of the operations provided by the `Frag` is devoted to the management of the shapes detected within the frag. Such operations include: associating different shapes to the frag, fetching the largest shape of a particular type, etc. As mentioned earlier in the present section, the `Frag` class is ultimately delegated to the execution of the recognition scans. This is the reason why the `Frag` class, unlike the `Slice` or the `Text` class, does not feature methods such as `invokeFirstScan()`, instead, it provides `executeFirstScan()`, `executeSecondScan()` and `executeThirdScan()`. The name of such operations is there to clarify that the actual evaluations are carried out within the frag.

The `Glyph` class comes into play during the second recognition scan, as the SW-OGR Engine infers the OGR area of the frag. Before that moment, it simply makes no sense to work with glyphs, since the system has no clue about the relationship between frags and glyphs. Discovering the OGR area of each frag sheds a fair light upon such relationship, thus the SW-OGR begins to work with glyphs. Any instance of the `Glyph` class carries the following information:

- A reference to the parent `Slice` object.
- A reference to the associated `Frag` object(s).
- The image of the glyph.

- The inferred OGR area code of the glyph (*HaC*, *Co*, etc.) and an indicator of the reliability of such information. The evaluation of the reliability is based on the number of inference checks passed (and not passed) by the frag.
- The `Coding` object(s) associated to the glyph, which keeps track of its ISWA code.

It is important to notice that a relevant part of the information of the `Glyph` class is gathered by aggregating the data coming from its associated frag(s). For example, the top left point, which is useful to keep track of the location of the glyph within its parent slice, is nowhere to be found among the fields of the `Glyph` class. Such information is dynamically calculated by gathering information about the top left point of the frags associated to the glyph. A number of operations of the `Glyph` class are designed to manage the relationship with the frags, e.g. retrieving information by a particular frag, updating the glyph with the information contained in its associated frags, etc. The `Glyph` class also features operations to manage the `Coding` object(s) it contains. Finally, we mention the very simple `Coding` class, whose only purpose is to store information about the ISWA coding of a given glyph. The class simply contains any field necessary to handle the three ISWA codings (International SignWriting Alphabet 2008 (ISWA-2008), ISWA-2010 and ISWA-BIANCHINI) and the operations required to manage them. Please notice the `getImage()` operation, whose purpose is to use the ISWA coding to fetch the official ISWA image of the recognized glyph. Such function is critical to build the final output image.

### The util package

The present section provides an in-depth view of the `util` package. As mentioned before such package is basically a multi-purpose library whose functions are used by many different classes, in different steps of the recognition procedure. The diagrams included in this section do not report all the classes of the `util` package. Classes designed for low-level utility purposes, such as logging, image displaying, configuration of image import/export paths, etc. are not described in this section, because their implementation is trivial. Since there is little or no dependency relationships between the classes of the `util` package, and since they contain a remarkable number of operations, the classes are shown separately.

The `ImgUtil` class is the image utility library of the `util` package (Fig 6.4). The operations provided by such library can be divided into two subsets. One subset provides functionalities for the analysis of the image, the other one deals with image manipulation

Among the operations devoted to image analysis, it is worth mentioning the one for the detection of 8-connected foreground components (`findBlobs()`) and holes (`findHoles()`), which are useful during the image segmentation step, and during the final recognition step of the procedure. A fair number



Figure 6.4: Class diagram of the util package: image utility library.

of image analysis operations provide different evaluations of the foreground on sections of the image. As an example, the `findForeground_Rect()` operation returns any foreground point within a given rectangle-shaped area of the image, similar functions are available for a number of other shapes. The image analysis subset also includes a number of other operations, such as `checkCircleSectorsCoverage()`, whose purpose is to assess the reliability of a detected circle by evaluating the foreground pixels around its circumference (see Borgia (2013) for a discussion).

Among the operations devoted to image manipulation, it is worth mentioning the implementation of the Zhang-Suen algorithm (T. Zhang & Suen, 1984) for image thinning by the graduating student Luca Franciosi (Franciosi, 2014), who took part in the development phase of the SW-OGR project. Such function also takes advantage of image pre-processing, i.e. hole removal, smoothing and acute angle emphasis (Stentiford & Mortimer, 1983), as well as post-processing procedures, i.e. Holt staircase removal (Holt, Stewart, Clint, & Perrott, 1987), which have been observed to improve the result of the thinning, as suggested in Parker (2010). The `ImgUtil` class also provides other manipulation functions, such as image rotation, color inversion, background filling, etc.

The `GeomUtil` class is the geometric utility library of the util package (Fig. 6.5). Most of the library is composed by operations on coordinates, which typically represent single points or shapes. Such operation include:

- Conversion between different coordinate systems (i.e. Cartesian coordinates and image coordinates).
- Different measurements over a set of coordinates, such as distance,



Figure 6.5: Class diagram of the util package: geometric utility library.

centroid, topmost point, etc.

- Detection of the direction of a point with respect to a reference point or shape, based on (Papadias, Sellis, Theodoridis, & Egenhofer, 1995).
- Manipulation of points or shapes, such as shifting, scaling, rotating, etc.
- Check of overlap and/or inclusion check of points and/or shapes.
- Angle detection, and other angle-related operations, such as `arctangent2()`

Please notice the `withinProjection()` operation in the diagram in Fig. 6.5. It is the implementation of the *inprojection* evaluation introduced in section 5.3.2, which takes place during the final recognition step of glyphs belonging to the Hand Configurations area. The last libraries that we take into account



Figure 6.6: Class diagram of the `util` package: math utility library and comparison library.

are the `MathUtil` and the `CmpUtil` classes (Fig 6.6). The first one provides low-level mathematical functionalities, which mainly consist in a wide range of operations on data arrays, such as smoothing, counting, filtering, etc. Fig. 6.6 also shows the `CmpUtil` class, which is a small library which provides comparators working on different data, mainly for sorting purposes.

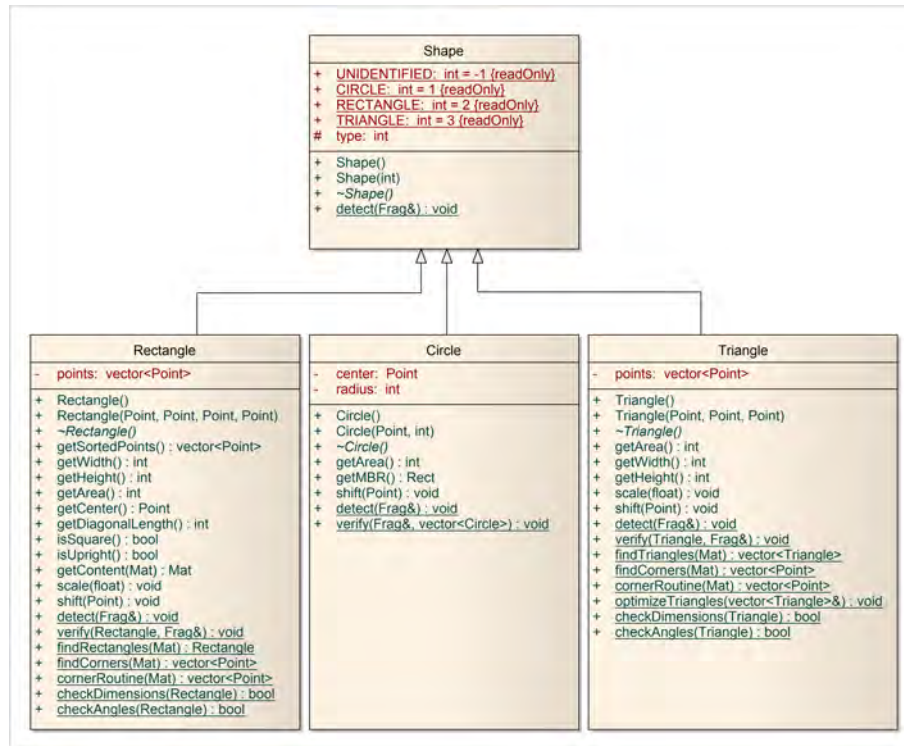


Figure 6.7: Class diagram of the `util` package: shape classes.

The description of the `Shape` superclass and its specialized classes in Fig. 6.7 concludes our analysis of the `util` package. The diagram in Fig. 6.7 only contains a restricted number of elements. Actually, we designed a subclass of `Shape` for each shape that the SW-OGR Engine needs to detect within a `SignWriting` text.

The `Shape` superclass is a very simple one, it contains one single field to keep track of the type of the shape (rectangle, circle, etc.). Except for constructors, getters and setters, the superclass only provides one single operation, i.e. `detect()`. Any subclass of `Shape` overrides such operation to provide detection functionalities. For example, the `detect()` operation provided by the `Rectangle` class is designed to analyze an input frag in order to detect the presence of rectangles.

Analyzing the subclasses, it is easy to see that any specialized shape adds its custom fields to those of the superclass. The `Circle` class, for example, stores the center of the circle and its radius, while the `Rectangle` class adds the coordinates of its corners. Each subclass also includes its own methods. A number of methods provide different evaluations on the shape, such as `getArea()` and `getWidth()`, while other methods allow the manipulation of the shape, such as `scale()` and `shift()`. As mentioned before, each subclass overrides the `detect()` method, however, shape detection can be a very tricky task, especially when dealing with handwritten symbols. For such reason, each detection routine may need a set of supporting operations. The



rectangle detection, for example, takes advantage of the corner detection performed within the `findCorners()` operation, to locate the vertexes of the possible rectangles within the image (see Section 6.2.3 for more details). Moreover, some types of shapes may require a verification step, after the detection process; this is the reason why the subclasses represented in Fig. 6.7 show operations like `verify()`, `checkAngles()`, etc. Those operations are invoked after the detection step, to check if the detected shapes are both reliable (i.e. making sure that the shape is actually there) and acceptable (i.e. making sure that the shape meets a number of requirements).

### The model package

The classes belonging to the `model` package are employed during the second recognition scan. During this scan, the SW-OGR Engine produces the OGR area inference for each frag, exploiting a wide range of information. The class diagram of the package is shown in Fig. 6.8. As mentioned in Section 6.1.2, the `model` package contains a class for each OGR area. Only two of them are shown in the class diagram, since their structure is very similar, and they present little or no mutual relationship.

The `Model` superclass and its subclasses are the only classes in this package.

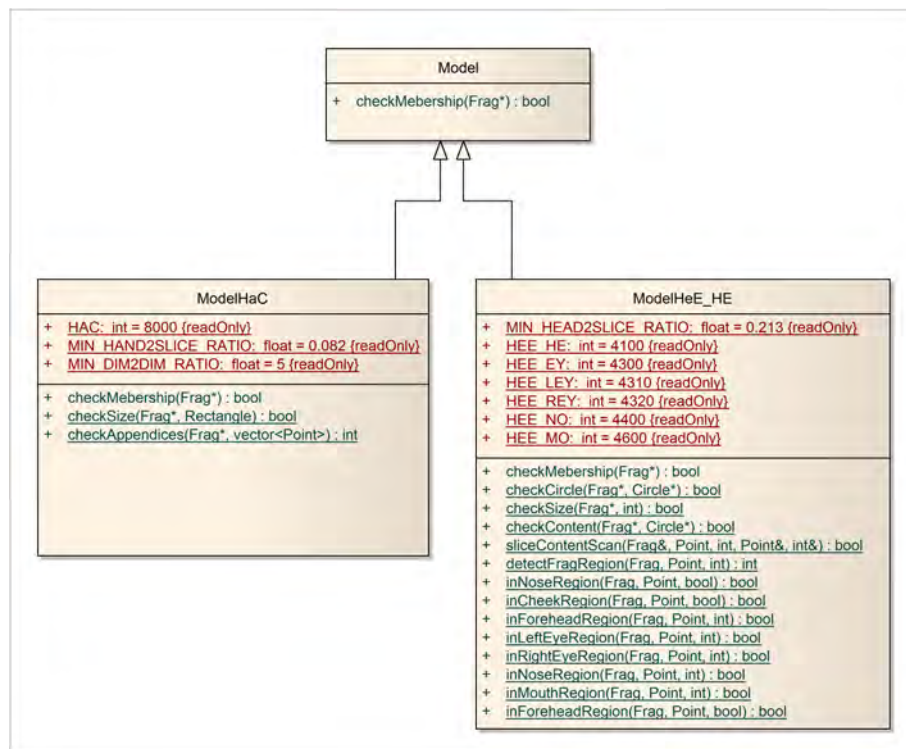


Figure 6.8: Class diagram of the `model` package.

Each subclass operates independently from the other subclasses, and they all specify the `Model` superclass, overriding its only provided operation, i.e.

`checkMemberShip()`. The purpose of the `checkMemberShip()` method of a given subclass `ModelXXX` is to analyze an input frag, in order to infer if it belongs to the OGR area that subclass `ModelXXX` represents. For example, the `checkMemberShip()` method provided by the `ModelHaC()` class evaluates the input frag, checking its size, and the presence of possible appendices (fingers) to assess if the frag may belong to the Hands Configuration Area. Each subclass provides a number of operations, such as `checkSize()`, to support the inference of the OGR area. Each class also has a number of constants carrying statistical data, to support the inference. As an example, the purpose of the `ModelHeE_HE` subclass is to understand whether a frag belongs to the Head Circles and Contacts area. Frags belonging to such area typically contain circles, just like those of other OGR areas. Their circles, however, are particular, since they are *bigger* than the others. The class constant `HEE_HEAD2SLICE_RATIO` helps in characterizing *bigger* by providing the usual ratio between the diameter of the circle, and the width of the slice. Such ratio is obtained by gathering statistical data out of our datasets, i.e. calculating the mean of the diameter-to-slice-width ratio over hundreds of head circles.

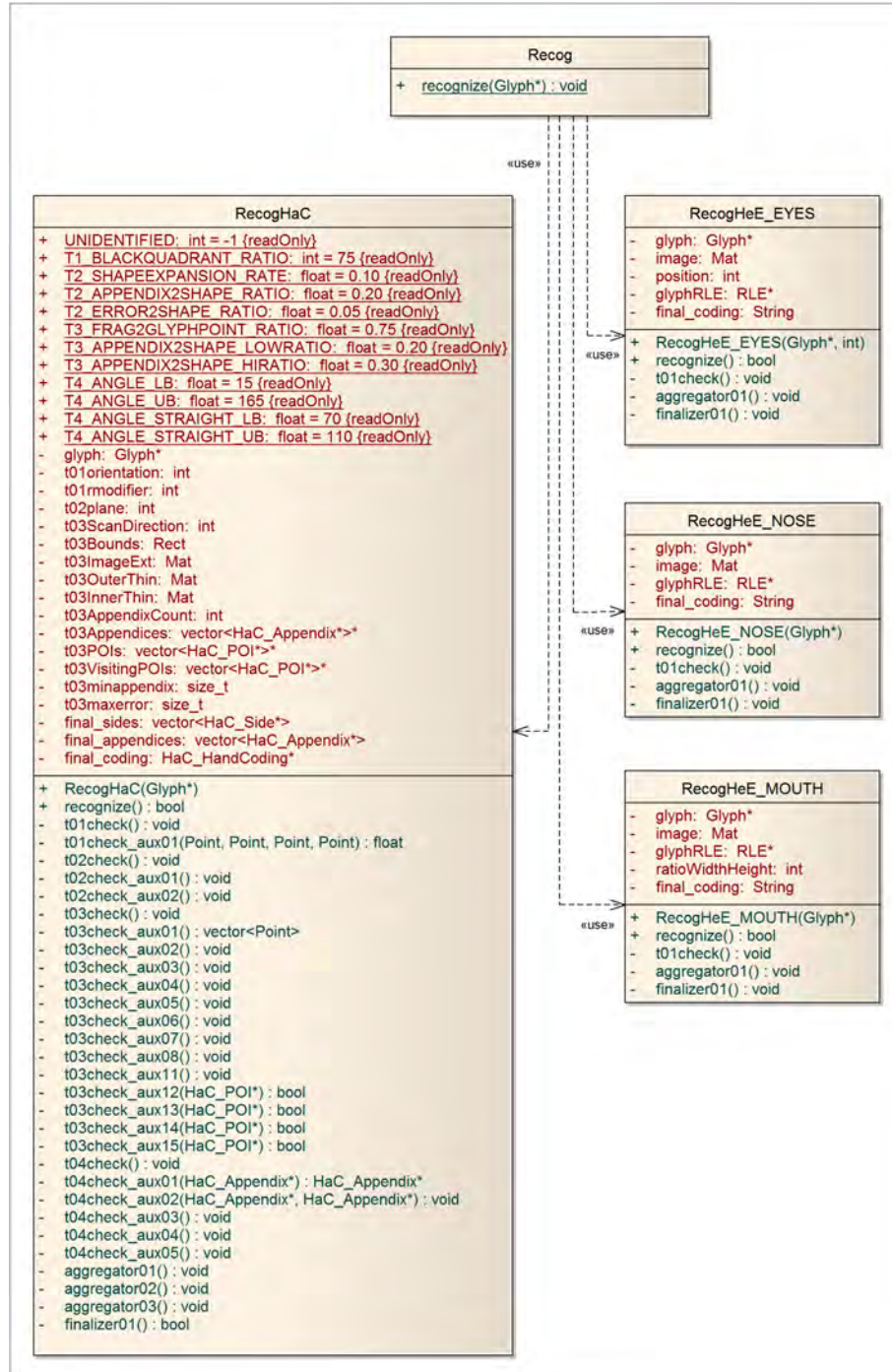
It is important to spend a few words about the `ModelHeE_HE` class, which represents an exception within the `model` package. A clue of this peculiarity can be found by observing the high amount of operations and methods that this class provides. In fact, the purpose of the `ModelHeE_HE` class is not only to assess if a frag is basically a head circle, but also to find any frag within that head circle, and to assign them to their respective area. For example, if a frag is found within the lower part of the head circle, it could likely be a frag belonging to the Facial Expressions: Mouth area. This is how the OGR area inference is performed for a large part of facial expressions.

If a class of the `model` package completes the OGR area inference procedure with success, a `Glyph` object is created and associated both to `Frag` and to the `Slice` objects. From this moment on, the SW-OGR Engine mainly works on glyphs (rather than frags). In fact, during the third recognition scan, frags are only used for low-level evaluations, while most of the work is performed on glyphs. Otherwise, when the reliability of the inference (introduced in Section 6.1.2) to a given OGR area does not exceed a fixed threshold value, the inference is to be considered failed. In other words, the analyzed frag does not represent any glyph belonging to that given OGR area. In this case, no glyph is associated to the frag, and the other classes of the `model` package execute their `checkMemberShip()` method on the frag. If no inference is possible, the frag is excluded from the third and final step of the recognition procedure. It will be processed in the final human-assisted review.

### The recog package

The classes belonging to the `recog` package are employed during the third recognition scan. During this scan, the SW-OGR Engine identifies the ISWA coding of each recognized glyph within a `SignWriting` text. Observing the



Figure 6.9: Class diagram of the `recog` package.

class diagram in Fig. 6.9), it is easy to see that the structure of the `recog` package is very similar to the one of the `model` package. There is a class for each OGR area, whose purpose is to carry out the recognition of any glyph belonging to that particular OGR area. Such classes do not share any relationship, except for the one with the `Recog` class. Such relationship, however, is a use relationship: this simply means that the `Recog` class makes use of the operations provided by the other classes. More specifically, the `Recog` class analyzes the inferred OGR area of the input glyph, in order to delegate its recognition to the most appropriate class. For example, if the glyph has been inferred to belong to the Hands Configuration area, the `Recog` class delegates the recognition to the `RecogHaC` class.

Except for constructors, getters and setters, the operations provided by each recognition class are divided into four important subsets.

- The *master recognition operation* (`recognize()`), which coordinates the whole recognition process for the input glyph by invoking the other operations.
- The *check operations* (`t01check()`, `t02check()`, ...), whose purpose is to perform different evaluations over the image of the glyph, in order to gather the information required to build the OGR code. Please notice that such evaluations can be very complex, so each check may require a number of support operations. Such operations are visible in Fig. 6.9, their names have the form `t01check_aux01()`, `t01check_aux02()`, etc. depending on the check operation they support.
- The *aggregators* (`aggregator01()`, `aggregator02()`, ...), whose purpose is to aggregate the information gathered by the check operations in order to produce the OGR code of the glyph. In other words, they process the results of the evaluations, which typically come in a wide variety of data types (booleans, vectors, etc.) and they generate a string containing the actual OGR code of the glyph.
- The *finalizer* (`finalizer01()`), whose purpose is to access the OGR-to-ISWA mapping tables (see Section 5.4) and to convert the OGR code of the glyph into the associated ISWA code.

As explained in Chapter 5, large and complex OGR areas (such as Hand Configurations) require longer OGR codes with respect to other, simpler areas (such as Facial Expressions: Mouth). Therefore, when dealing with a large OGR area, it is necessary to perform a large number of checks over the image of the glyph, thus the recognizer class carries a larger number of field and operations. This explains the remarkable difference, visible in Fig. 6.9, between the `RecogHaC` class, and a simpler one, such as `RecogHeE_MO`. Finally, it is worth reminding that, since the `recog` package contains a high number of classes, and since their structure is very similar, Fig. 6.9 shows only a restricted number of elements.

## 6.2 Design and development of SW-OGR

The present section provides an in-depth view of the development of the recognition procedure implemented by the SW-OGR Engine. Since the application features a fairly high amount of code (more than 17,000 lines, so far), this section reports only the development details that may be of interest from the point of view of computer science researchers and computer vision experts.

The SW-OGR Engine is currently developed as a C++ application. The application takes advantage of the OpenCV (Open Source Computer Vision) library (Itseez, 2014), which is an open source computer vision and machine learning software library. According to its developers (Itseez, 2005), OpenCV is “built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which include a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms”. Moreover, the application takes advantage of the Boost C++ Libraries (*Boost C++ Libraries*, 2004; Nakariakov, 2013), a set of libraries for the C++ programming language that provide support for tasks and structures such as linear algebra, pseudo-random number generation, multithreading, image processing, regular expressions, and unit testing. Boost C++ Libraries were mainly included within our application to support the use of regular expressions, especially when dealing with the OGR to ISWA mapping tables.

### 6.2.1 Image pre-processing

As mentioned very often in the present chapter, the SW-OGR Engine deals with the recognition of symbols, and the pre-processing and binarization steps are the earliest ones of the recognition procedure. In such early steps, there is actually no difference between OGR and OCR, so we took advantage of the rich image processing literature about OCR, focusing on these operations.

The presence of noise within digital images is a very common problem, and the countermeasures for this are well consolidated in computer science literature. More specifically, when dealing with the recognition of symbols, it is advisable to apply noise reduction before thresholding, so that the filters devoted to noise reduction can exploit a more reliable information. A possible approach to noise reduction, when dealing with a paper source (Parker, 2010), requires acquiring multiple images of the same page. “Averaging the grey levels of each pixel across all the samples will give a much better result as far as noise is concerned. Averaging four samples, for examples, cuts the noise in half”. Of course, this is a long process, and very often it is not possible to have the source paper document at hand, to perform multiple scans. In such cases, one of the best options (Gonzalez & Woods, 2011; Parker, 2010) is to blur the image by applying a *median filter*. A median

filter is basically a pass through all pixels of the image, considering an  $n \times m$  window centered at each pixel. The pixel in the center is replaced by the median value of all pixels in the window. Such procedure, already implemented within the OpenCV library (`cv::medianBlur()`), removed most noise within our source images.

## 6.2.2 Image binarization

In the image processing theory, segmentation is the generic process by which an image is divided into its homogeneous regions, with respect to a certain criterion (e.g. texture) or even into objects. Such division is typically carried out to an extent which is determined by the problem which is to be solved. In other words, the segmentation should stop when the objects of interest in an application have been isolated (Gonzalez & Woods, 2011).

Binarization, on the other hand, is usually carried out as a part of the segmentation process, and its purpose is to transform a grayscale (or BGR, CMYK, etc.) image into a binary one (black and white), typically to identify the foreground from the background. “A binary image should contain all the essential information concerning the number, position, and shape of objects while containing a lot less information. The essential reason for classifying pixels by grey level is that pixels with similar levels in a nearby region usually belong to the same object, and reducing the complexity of the data simplifies many recognition and classification procedures. Thresholding is almost essential before thinning, vectorization, and morphological operations” (Parker, 2010).

### 6.2.2.1 Thresholding methods

Please notice that SignWriting texts are acquired as grayscale images by the SW-OGR Engine. The most common method to perform binarization (Parker, 2010) is to select a single threshold value (thresholding). All the grey levels below this value will be classified as black (0), and those above will be white (1). “The segmentation problem becomes one of selecting the proper value for the threshold” (Parker, 2010). The choice of the threshold is usually performed by analyzing the image histogram. In the case of handwritten SignWriting documents, binarization identifies the symbols within the text from the background.

The texts within our datasets present remarkable differences in their histograms. As a consequence, a single fixed threshold value for all the images processed by the SW-OGR Engine was never an option. Threshold values that might seem appropriate for some texts, might make other texts completely unusable. Fig 6.10 show an example of the consequences of a fixed threshold value for all the texts. The three slices on the left part of the Fig 6.10 have all been binarized using 100 as threshold value (the binarized slices are visible on the right part of Fig 6.10). It is evident that the thresholding

result on the first slice is acceptable, while the same cannot be said for the other two slices.

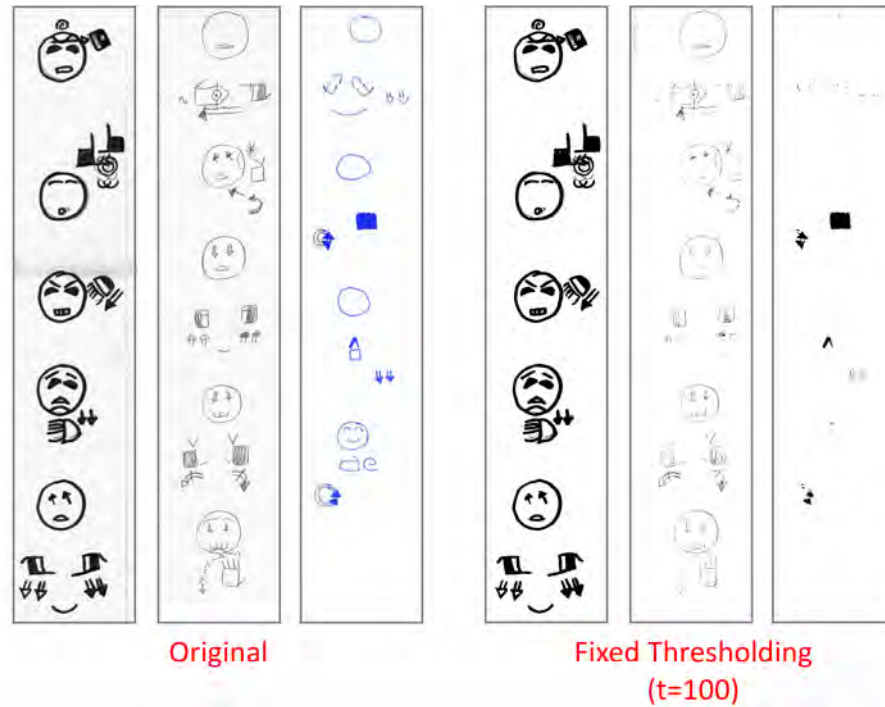


Figure 6.10: Fixed thresholding ( $t = 100$ ) applied to different slices.

### 6.2.2.2 Thresholding performance evaluation

With the help of graduating student Marco Rotiroti (Rotiroti, 2014), we applied a number of state-of-the-art thresholding methods available in image processing literature to our texts (Sezgin & Sankur, 2004), and we performed a statistical analysis in order to choose the best candidate. The images were processed with both *global* and *local* thresholding methods. Global thresholding methods aim at identifying a threshold value for the whole image and then use it to divide it into two classes: foreground and background. They typically perform the binarization process in a quite simple and fast way. As mentioned before, several methods for the determination of a threshold value are based on the histogram of the image. However, in the case of images with uneven illumination, the choice to perform a global thresholding might not be appropriate. Local thresholding can help solve this problem. In fact, local thresholding methods statistically determine the threshold values examining the intensity values within the neighborhood of each pixel (e.g. local average, variance, standard deviation, etc.). Compared with global thresholding methods, local ones are typically more accurate. However, from the point of view of performance, they are much more demanding. The following list

reports the methods used within the test. Please notice that the description of each method is out of the scope of the present work.

- Global thresholding
  - Basic thresholding (fixed,  $t=100$ ) (Sezgin & Sankur, 2004).
  - Basic thresholding ( $t$ =average intensity) (Sezgin & Sankur, 2004).
  - P-tile (Sezgin & Sankur, 2004).
  - Otsu (Otsu, 1979).
- Local thresholding
  - Niblack (Niblack, W., 1985).
  - Sauvola (Sauvola, J. and Pietikäinen, M., 2000).

Fig. 6.11 and Fig. 6.12 show the performance of the thresholding methods reported in the above list, applied to two of the slices belonging to the DS-PEAR dataset. Of course, the figures simply provide a qualitative idea about the outcome of the test. Despite this, it is already possible to notice the higher accuracy characterizing Otsu and Sauvola, with respect to the other methods.

**6.2.2.2.1 Testing methodology** In order to perform the quantitative analysis of the thresholding methods <sup>2</sup>, we picked 7 images (slices) from the DS-PEAR dataset, Fig. 6.11 and Fig. 6.12 show slice 1 and 4 respectively. The selection of the images was based on different factors, such as stroke, resolution, contrast, amount of symbols per slice and noise. Slice 1 and 4 are shown here to demonstrate that even a slight variation in the stroke of the symbols (heavier in Fig. 6.11 and lighter in Fig. 6.12) can dramatically affect the result of many thresholding methods. The performance evaluation of the different thresholding methods was mainly conducted considering the two metrics, *misclassification error* (ME) and *relative foreground area error* (RAE), described in the survey by Sezgin and Sankur (2004).

Both metrics, i.e. ME and RAE, require the use of a reference image (*ground truth* image). Such image represents the best result that can be expected as the outcome of the thresholding. During our test, we adopted two different methods (Sezgin & Sankur, 2004) for the creation of the ground truth image. The first one (GT1) is based on manual segmentation of the different images carried out by a group of image processing and SignWriting experts. The second one (GT-2) is based on the application of different thresholding methods over the same image, and evaluating the probability that a pixel

<sup>2</sup>In order to apply the p-tile method, it is mandatory to know *a priori* the percentage of the foreground pixels within the image. Such knowledge, i.e. the percentage of the text occupied by the symbols, is currently beyond reach. As a result, the p-tile method has not been included into the test. In order to generate Fig. 6.11 and 6.12, the p-tile method has been applied using default percentages commonly used in OCR (Parker, 2010).

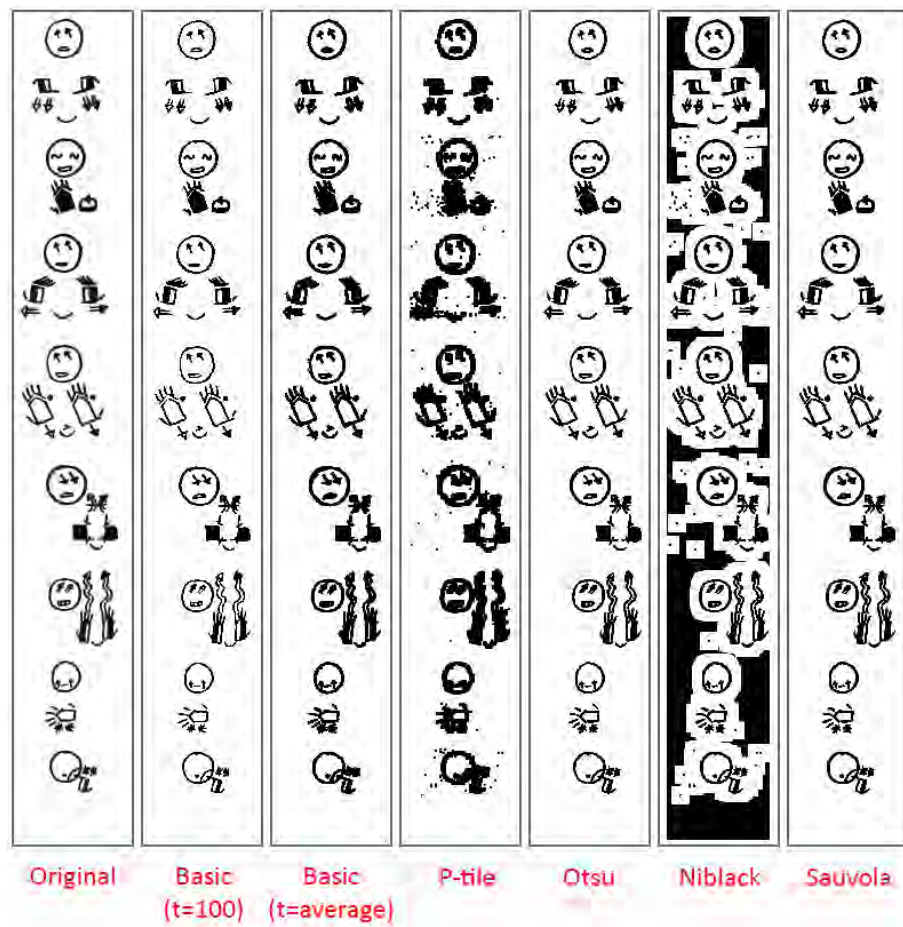


Figure 6.11: Multiple thresholding methods applied on the same slice: test slice 1.



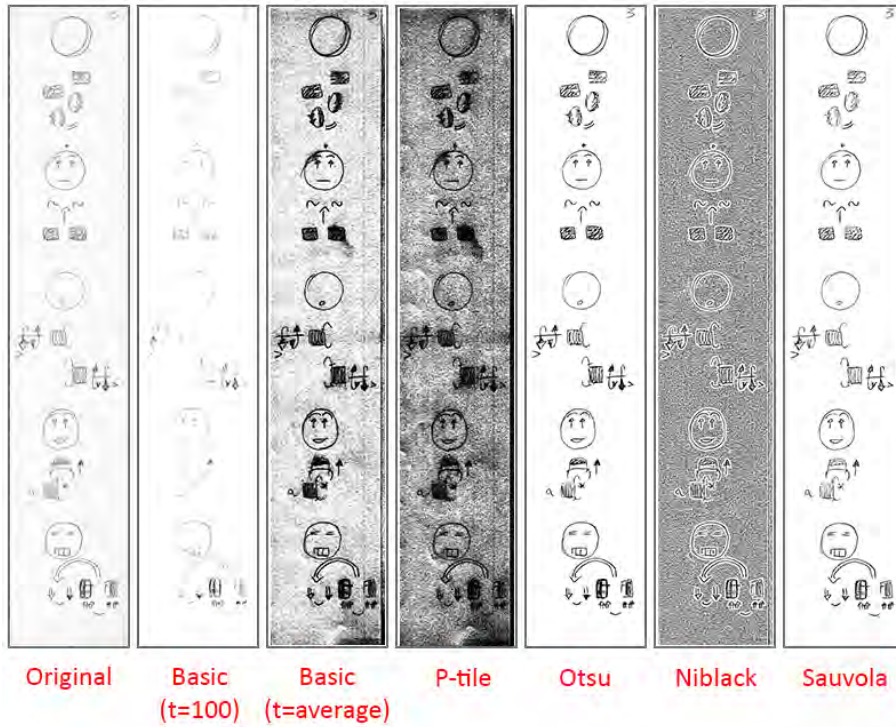


Figure 6.12: Multiple thresholding methods applied on the same slice: test slice 2.

is included within the foreground or not. More details about the creation of the ground truth images are provided in Rotiroti (2014).

The ME metric evaluates the percentage of background pixels wrongly assigned to the foreground and, vice versa, the pixels of the foreground erroneously assigned to the background. The ME can be defined as:

$$ME = 1 - \frac{|B_O \cap B_T| + |F_O \cap F_T|}{|B_O| + |F_O|}$$

where  $B_O$  and  $F_O$  indicate the background and the foreground of the reference image (or *ground truth* image),  $B_T$  and  $F_T$  denote the background and foreground in the image which is undergoing the test, and  $|\cdot|$  denotes the cardinality of the set. The outcome of the ME ranges from 0 (perfect) to 1 (completely incorrect).

The RAE metric is actually a comparison of different properties of the foreground, i.e. area of foreground detected and expected. It is defined by the following equation:

$$RAE = \begin{cases} \frac{A_O - A_T}{A_O}, & \text{if } A_T < A_O \\ \frac{A_T - A_O}{A_T}, & \text{if } A_T \geq A_O \end{cases}$$

where  $A_O$  is the area of the ground truth image and  $A_T$  is the area of the one which is undergoing the test. In case of a perfect correspondence of the regions, the RAE assumes the 0 value, while 1 represents a total lack of



overlap between the two areas.

Since we created two categories of ground truth images, we performed two series of tests. Each one was performed by analyzing the outcome of ME and RAE metric for each thresholding method. The first series of tests made use of ground truth images generated through the GT1 method, while GT2 was employed for the second series. For each series, the evaluation of the thresholding methods was carried out through the following steps:

1. Pick a thresholding method  $k$  belonging to the set  $S = \{T_{100}, T_{average}, \text{P-tile}, \text{Otsu}, \text{Niblack}, \text{Sauvola}\}$ :
2. For each slice  $i=1,2,\dots,7$ , thresholded with  $k$ , calculate the average  $\bar{\omega}(i)$  between the results of the  $ME(i)$  and the  $RAE(i)$ .

$$\bar{\omega}(i) = \frac{ME(i) + RAE(i)}{2}$$

3. Sum the  $\bar{\omega}(i)$  values of all slices  $i$  processed with the thresholding method  $k$ .

$$score(k) = \sum_{i=0}^7 \bar{\omega}(i)$$

4. The optimal thresholding method  $mth$  is the one which yields the lowest  $score(k)$ .

$$mth = \arg \min_{k \in S} score(k)$$

**6.2.2.2.2 Results and discussion** The results of the test with using GT1 and GT2 are rather similar. It is interesting, anyway, to report and discuss them separately, to have a more detailed view of our findings.

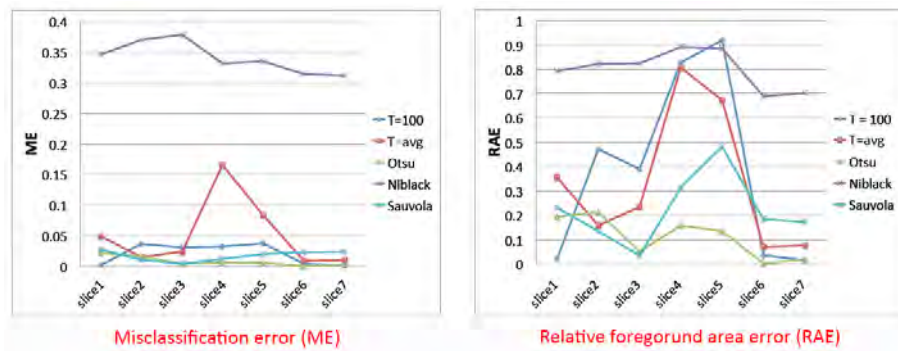


Figure 6.13: Test series (ground truth GT1): outcome of ME and RAE metrics for image thresholding evaluation.

As we can see in Fig. 6.13 and Tab. 6.1, the results of ME cannot by themselves identify a single thresholding method as the most appropriate one.

In fact, different methods, whether they are global (Otsu and  $T_{100}$ ) or local (Sauvola) have a minimum value below 0.05 for all seven images. As an example, the minimum values on slice 1 and 7, are 0.0017 and 0.0019 respectively, and they were obtained using the  $T_{100}$  method. Otsu resulted the most accurate method for slice 6 with a value of 0.0002. Finally, the minimum values on slice 2 and 3 were achieved by Sauvola, obtaining values equal to 0.0105 and 0.0039 respectively. The only methods that can be excluded without the further assessment by RAE are the global method based on the average value and the Niblack method. In fact, examining the slice 4 in Fig. 6.12, we can see how both methods introduce a considerable amount of salt and pepper noise.

#### Test series (ground truth GT1): misclassification error

Image	$T_{100}$	$T_{average}$	Otsu	Niblack	Sauvola
slice 1	<b>0.0017</b>	0.0495	0.0217	0.3461	0.0271
slice 2	0.0368	0.0146	0.0166	0.3708	<b>0.0105</b>
slice 3	0.0312	0.0244	0.0041	0.3791	<b>0.0039</b>
slice 4	0.0324	0.1654	<b>0.0062</b>	0.3319	0.0123
slice 5	0.0378	0.0839	<b>0.0056</b>	0.3352	0.0199
slice 6	0.0044	0.0091	<b>0.0002</b>	0.3148	0.0231
slice 7	<b>0.0019</b>	0.0100	0.0025	0.3116	0.0239

Table 6.1: Test series (ground truth GT1): misclassification error. Minimum (i.e. optimal) values for each slice are highlighted.

Performing a quantitative analysis of the results of RAE, it is possible to notice (Tab. 6.2), that the  $T_{100}$  method still produces the best result for the slice 1 and 7. However, as the contrast of the slices decreases (slice 2-5, see Fig. 6.12), the histograms assume a unimodal distribution with negative asymmetry (*skewness*). In such cases, the  $T_{100}$  method behaves worse than  $T_{average}$  and Niblack, that were previously discarded. The method by Otsu presents, in some cases, values that exceed the  $T_{100}$  method (slice 1 and 7), the  $T_{average}$  method (slice 2) and the Sauvola method (slice 2-3), but it appears to be the one with the greater number of slices (5 out of 7) segmented with a RAE value below 0.2 (for slice 4 and 5 Otsu is the only one below this threshold).

It is important to notice that the selection of a thresholding method for the SW-OGR Engine using the test series with GT1 may be compromised by the incorrect creation of the reference images. In fact, the ground truth images by GT1 method are created by expert users, i.e. human beings, who may be inaccurate, in some cases. This causes the presence of less reliable values in both ME and RAE. To avoid such bias, it is important to also perform a quantitative analysis on the data coming from the test series performed with GT2.

**Test series (ground truth GT1): relative foreground area error**

Image	$T_{100}$	$T_{average}$	Otsu	Niblack	Sauvola
slice 1	<b>0.0191</b>	0.3557	0.1952	0.7942	0.2314
slice 2	0.4719	0.1580	0.2125	0.8222	<b>0.1345</b>
slice 3	0.3902	0.2339	0.0510	0.8251	<b>0.0355</b>
slice 4	0.8284	0.8090	<b>0.1587</b>	0.8913	0.3142
slice 5	0.9197	0.6709	<b>0.1354</b>	0.8868	0.4832
slice 6	0.0361	0.0693	<b>0.0022</b>	0.6892	0.1847
slice 7	<b>0.0159</b>	0.0765	0.0205	0.7001	0.1727

Table 6.2: Test series (ground truth GT1): relative foreground area error. Minimum (i.e. optimal) values for each slice are highlighted.

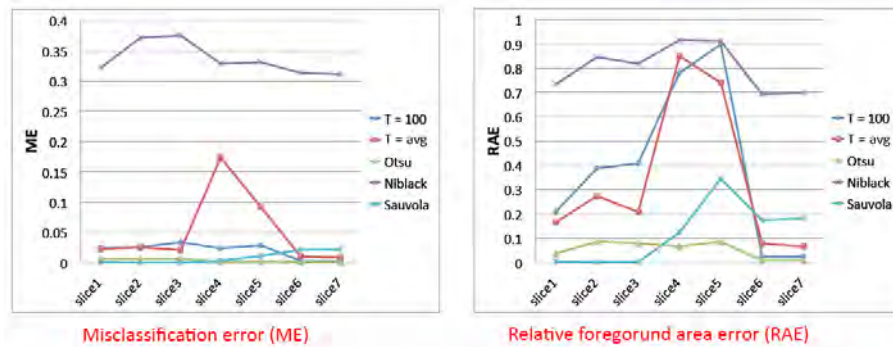


Figure 6.14: Test series (ground truth GT2): outcome of ME and RAE metrics for image thresholding evaluation.

The graph in Fig. 6.14 summarizes the results of the performance test series performed using GT2. Analyzing the graph, it is evident that the optimal results of ME and RAE obtained by method  $T_{100}$  on slice 1 and 7 when comparing with GT1, are replaced by those produced by Sauvola (slice 1) and Otsu (slice 7).

More specifically, performing a quantitative analysis of the results of ME (Tab. 6.3), we notice that Sauvola presents values below 0.0010 for slices 1-3, while Otsu scores below 0.0030 for the remaining slices. Since the generation of the ground truth image is not influenced by considerations made by users, the analysis of the ME proves to be more accurate. Despite this, the selection of a single thresholding method is not yet possible without evaluating the results of the RAE along with those of ME.

**Test series (ground truth GT2): misclassification error**

Image	$T_{100}$	$T_{average}$	Otsu	Niblack	Sauvola
slice 1	0.0245	0.0232	0.0067	0.3221	<b>0.0008</b>
slice 2	0.0261	0.0253	0.0068	0.3721	<b>0.0002</b>
slice 3	0.0338	0.0218	0.0070	0.3751	<b>0.0002</b>
slice 4	0.0239	0.1739	<b>0.0022</b>	0.3288	0.0038
slice 5	0.0291	0.0925	<b>0.0031</b>	0.3311	0.0112
slice 6	0.0032	0.0104	<b>0.0017</b>	0.3133	0.0216
slice 7	<b>0.0031</b>	0.0088	0.0015	0.3113	0.0222

Table 6.3: Test series (ground truth GT2): misclassification error. Minimum (i.e. optimal) values for each slice are highlighted.

Analyzing the graph (Fig. 6.14) and the values (Tab. 6.4) of the RAE metric, it is possible to notice that the best score on slices 1-3 is achieved by Sauvola, whether they present a slightly bimodal histogram (slice 1) or a reduced contrast (slice 2 and 3). However, considering the results of the RAE scored by Sauvola, it is possible to notice the value 0.3449 for slice 5. This latter value implies a number of problems related to the presence of areas with reduced contrast, which is distinctive of images featuring a high concentration of dark pixels. Summarizing the above results, we can say that the  $T_{100}$ ,  $T_{average}$  and Niblack methods do not allow to perform the binarization process for the entire dataset, since they do not score any minimum (optimal) for both ME and RAE. This is due to issues such as the introduction of post-thresholding salt and pepper noise ( $T_{average}$ ), the persistence of background noise (Niblack) or a completely inadequate (too low) choice of thresholding value ( $T_{100}$ ). The GT1 graph in Fig. 6.13 is characterized by the presence of sharp peaks in the results of the RAE for all thresholding methods, while the GT2 shows that a single method, (i.e. Otsu) is the only one featuring an almost “linear” trend.

Finally, it is interesting to compare the results of both test series GT1 and GT2. Fig. 6.15 shows the overall performance of the different thresholding methods calculated for both GT1 and GT2. It is important to notice that in both graphs, the Otsu method appears to be the one with the least (optimal) score. As a consequence, the thresholding method we chose to use in the binarization step of the SW-OGR Engine is Otsu. Such method is already implemented within the OpenCV library (`cv::threshold()`).

#### Test series (ground truth GT2): relative foreground area error

Image	$T_{100}$	$T_{average}$	Otsu	Niblack	Sauvola
slice 1	0.2117	0.1668	0.0392	0.7340	<b>0.0061</b>
slice 2	0.3882	0.2733	0.0876	0.8466	<b>0.0028</b>
slice 3	0.4098	0.2086	0.0815	0.8194	<b>0.0036</b>
slice 4	0.7812	0.8502	<b>0.0678</b>	0.9148	0.1256

*Continued on next page*

**Test series (ground truth GT2): relative foreground area error –**  
*Continued from previous page*

Image	$T_{100}$	$T_{average}$	Otsu	Niblack	Sauvola
slice 5	0.8983	0.7405	<b>0.0876</b>	0.9108	0.3449
slice 6	0.0262	0.0788	<b>0.0124</b>	0.6924	0.1763
slice 7	0.0255	0.0675	<b>0.0110</b>	0.6972	0.1807

Table 6.4: Test series (ground truth GT2): relative foreground area error. Minimum (i.e. optimal) values for each slice are highlighted.

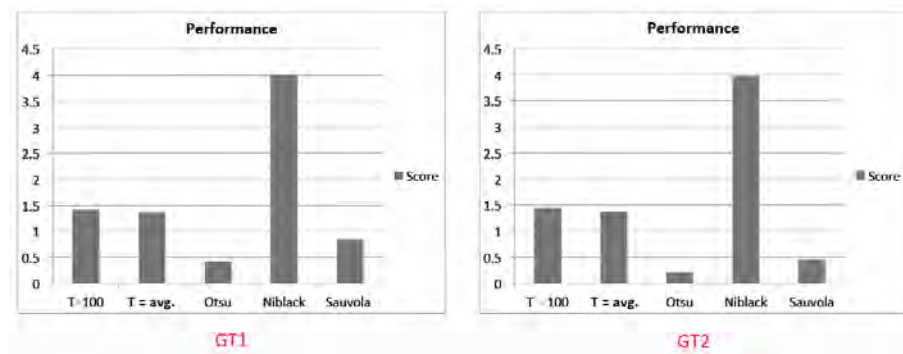


Figure 6.15: Final performance results for the thresholding evaluation test.

## 6.2.3 Shape detection

The purpose of the first step of the actual recognition procedure is to detect the presence of a number of base shapes within the text. Such detection is of critical importance for the following recognition steps, and it is performed frag by frag. The present section provides an in-depth view of a number of relevant shape detection procedures adopted by the SW-OGR Engine.

### 6.2.3.1 Circles

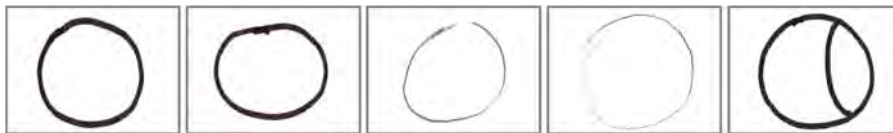
There is a lot of stories about the famous painter Giotto. One of them is particularly interesting, in our case.

*Pope Boniface VIII wanted to commission some paintings for St. Peter's and so he sent a courtier around to find the best painter in Italy. The courtier asked all the artists to give him a sample of their work to send to the Pope. He came to Giotto's workshop, explained his mission, and asked him for a drawing which would give the Pope some idea of his competence and style. "Sure", said Giotto; and he laid down a sheet of paper, reached for a brush dipped in red paint, closed his arm to his side to make a sort of compass of it, and in one even sweep scribed a perfect*

*circle. “There you are”, he told the courtier, handing it to him with a smile.*

Unlike Giotto, handwriting a perfect circle is beyond the reach of most human beings, and SignWriting users make no exception. Moreover, the production of a handwritten text is generally a fast, efficient process, which aims at producing the required amount of symbols in the shortest time possible. Taking the time to handwrite a perfect circle every time that a head or movement glyph is required is not efficient at all. Therefore, when dealing with SignWriting texts, it is very frequent to find “circles” which are drawn in a very inaccurate way. Fig. 6.16 reports a few examples of circles, extracted from our datasets (DS-PEAR and DS-POTIERS). The symbols present a different level of accuracy.

Two options have been explored in order to perform a fast and reliable detection of the circles within a frag. The first one is based on the detection of holes within the image (Wang, 1998). Such approach implies searching for all the holes (i.e. a background area enclosed within a foreground area) and evaluating their shape and dimensions to check if any of them can be assimilated to a circular shape. It is important to remark that this technique is not mainly intended for circle detection, and it is not a recent technique. Despite this, it has two important advantages: first of all, it is very efficient: its time complexity is  $O(n)$  (where  $n$  is the number of pixel within the image). Moreover, the symbols of SignWriting featuring holes are distributed among a very small number of OGR areas. We observed that simple analyses over the size and the shape of the holes can help in understanding the OGR area of the frag. However, such method present different drawbacks. One above all is that, when dealing with handwritten symbols, it is common to find circles which have not been completely drawn, and present an incomplete circumference (see Fig. 6.16 for an example). In such cases, no hole can be identified within the image of the frag. The circumference could, in some cases, be forcibly closed by applying morphological operators (Parker, 2010), but the image would be unpredictably altered. Such option was therefore discarded.



*Figure 6.16:* A number of circles, extracted from both DS-PEAR and DS-POITIERS datasets.

The second option is a very popular one in image processing theory, and it is the one that is currently implemented within the SW-OGR Engine. Such option is based on the Hough transform (Hough, 1959), more specifically it exploits the generalized Hough transform for the detection of curves (Duda

& Hart, 1972). The generalized Hough transform performs very well with most curves within our SignWriting texts, even if they are drawn very inaccurately. The OpenCV implementation of the transform (`cv::houghCircles()`) also allows to specify a number of handy parameters to filter the outcome of the transform, such as the minimum radius of the detected circles, the minimum distance between the center of two detected circles, etc. Fig. 6.17 shows the results of the application of the generalized Hough transform for the detection of curves to the circles which have already been introduced in Fig. 6.16.



*Figure 6.17:* Application of the generalized Hough transform for the detection of curves to images which contain circles.

However, the generalized Hough transform presents an important drawback too. The ISWA, in fact, contains many symbols which do not have a circular shape, but are composed of curved lines. Many of them, for example, are gathered in the Hand Movements area. Such symbols are identified as (parts of) circles too (see Fig. 6.18), and this may cause major inaccuracies to the whole recognition procedure. To solve such issue, we devised (Borgia, 2013) a simple circle reliability check, whose purpose is assessing if the circle detected by the generalized Hough transform for curve detection is actually a circle. The circle detection reliability check is described in Alg. 1, and illustrated in



*Figure 6.18:* Application of the generalized Hough transform for the detection of curves to images which do not contain circles.

Fig. 6.19. The concept is very simple: the algorithm checks for the presence of foreground points “near” the circumference of the detected circle, to assess if it is actually a circle, or just a curve. The analysis of the foreground points near the circumference is carried out by identifying a circle ring around the circumference (see Fig. 6.19). The circle ring is split into a number of sectors (the smaller the angle of the sectors, the more accurate the reliability check). The algorithm checks for the presence of foreground pixels within each sector. If the number of foreground pixels is greater than a threshold

value, the sector is marked. The threshold value (FPt in Alg. 1) is set to half the length of the circumference sector. Once all sectors have been analyzed, the percentage of marked sectors is used to evaluate the reliability of the circle detection. If such percentage exceeds 75% the detected shape is most likely a circle. Else, if the percentage of marked sectors only exceeds 50%, the shape is a curve. Otherwise, the detected shape is just a section of a curve. Please notice that each of those 3 possibilities is a valuable information, and it is exploited during the inference step of the detection procedure, to determine which OGR area the frag may belong to.

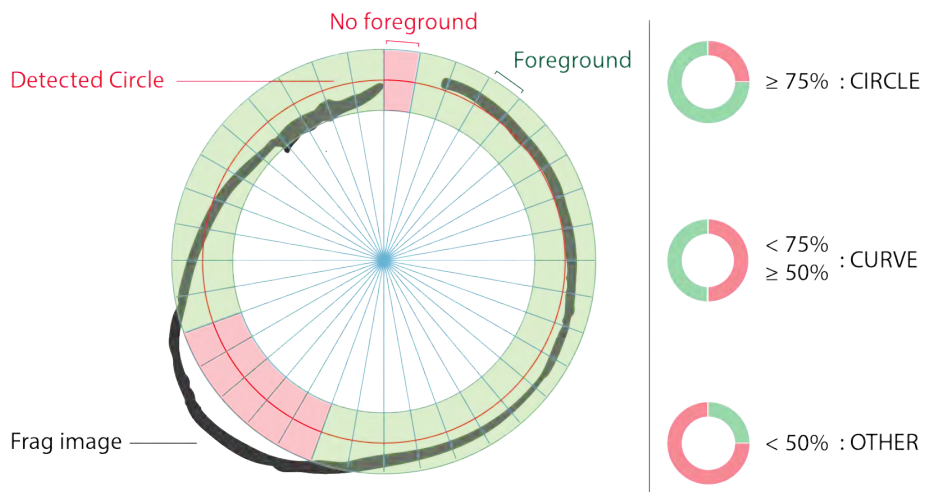


Figure 6.19: Illustration of the circle detection reliability check implemented by the SW-OGR Engine.



**Algorithm:** CircleDetectionReliability**Data:**I: the **image** of the frag.C: the **circle** detected (generalized Hough transform) within the frag.

Ang (integer) : 5.

Rng (float):  $0.10 * C.radius$ .FPt (float):  $(C.circumference / (360 / Ang)) * 0.5$ .**Result:**

1, if the detected circle is actually a circle.

2, if the detected circle is actually a curve.

-1, otherwise.

Let  $Sec = 0$  be a numeric constant such that Let CMin and CMax be two *circles*;

CMin.center = C.center; CMin.radius =  $(C.radius - Rng)$ ;CMax.center = C.center; CMax.radius =  $(C.radius + Rng)$ ;Let R be the **circle ring** obtained by CMax - CMin;Let  $RS = \{S_1, S_2, \dots, S_n\}$  be the set of **ring sectors** of R with an angle of Ang degrees;**for** each  $S_i$  in  $RS$  **do**

```

|   if  $S_i$  contains more than FPt foreground pixels: then
|   |   Sec += 1;

```

**end****if**  $Sec \geq 0.75 * RS.size$  **then**

```

|   /* 75% sectors covered, the shape is a circle. */
|   return 1;

```

**else if**  $Sec \geq 0.50 * RS.size$  **then**

```

|   /* 50% sectors covered, the shape is a curve. */
|   return 2;

```

**else**

```

|   return -1;

```

**Algorithm 1:** Circle reliability detection function implemented within the SW-OGR Engine.

### 6.2.3.2 Rectangles, triangles and other polygons

The observation about the inaccuracy of handwriting does not hold for circular shapes only, but also for the other shapes that SignWriting users draw when producing a text. Polygons such as rectangles, triangles, etc. make no exception. The present section describes the polygon detection algorithm implemented within the SW-OGR Engine. The rectangular shapes, i.e. the most common shapes featured in SignWriting symbols, will be used as an example in the present section to illustrate the detection algorithm. Anyway, the described approach is the same for any other polygon.

We explored two possible ways of detecting polygons: the first one relies on the Hough transform for line detection (Hough, 1959), more specifically, we tested both the standard and the probabilistic Hough transform (Duda & Hart, 1972). Both methods are implemented within the OpenCV libraries (`cv::houghLines()` and `cv::houghLinesP()`). In the field of computer vision, such methods are very useful to detect polygons within an image, since the detected lines carry valuable information about the location of the foreground pixels. The lines are typically processed and grouped according to their length, start and end points in order to build polygons, in a very fast and efficient way. However, after a number of tests on both the DS-PEAR and the DS-POITIERS datasets, we discarded both the Hough methods. The results of the line detection were mainly reliable in the presence of well-drawn, straight lines. We observed both methods failing, with different degrees of severity, in the following cases:

- Imperfections in the lines of the polygons (staircases, foreground clusters, etc.).
- Images with very low resolution.
- Polygons featuring appendices stemming from sides and/or corners (very common in SignWriting).

The second polygon detection algorithm we tested was far more reliable, and it is currently in use within the SW-OGR Engine. The algorithm basically looks for those points of the frag which could be the corners of the polygon, in our case, the rectangle. After this, the algorithm tries to build a polygon with the detected points. The  $K$  polygons with the largest area are chosen among all others and further analyzed.

We tested two corner detection routines, i.e. the Harris method (Harris & Stephens, 1988) and the Shi and Tomasi method (Shi & Tomasi, 1994), also known as Good Features To Track (GFTT). Both methods are implemented within the OpenCV library (`cv::cornerHarris()` and `cv::goodFeaturesToTrack()`). The performances of the two methods are very similar, when dealing with binary images. Anyway, we decided to adopt the Shi and Tomasi method since its OpenCV implementation allows a better filtering of the detection results, increasing the overall efficiency of the algorithm. The upper part of Fig. 6.20 shows the results of the Shi and Tomasi method, applied to a

number of rectangles. The detected features are represented as small red circles over the original image.

During the second part, each detected corner  $C$  is processed separately: first of all, the algorithm finds the set of detected corners which are *connected* to  $C$ . For the sake of clarity, a corner  $D$  is connected to  $C$  if the straight line between  $C$  and  $D$  is exclusively composed by foreground pixels (with an error margin). After this, the algorithm tries to build a polygon, in our case a rectangle, using the set of connected corners and evaluating their relative distance. The detected polygons are stored within a container, and sorted by their area, in descending order. The  $K$  polygons with the largest area are picked as the result of the detection. In most cases,  $K$  is equal to 1. In other cases, such as the Hand Movements area, the glyphs may feature more than one polygon (triangles, in particular), so it is useful to keep track of more than 1 of them; the recognition step for such areas is implemented taking into account this particularity. The algorithm is also capable of generating polygons starting from an insufficient number of connected corners. As an example, it is able to build a rectangle, starting from three connected corners, and guessing the position of the missing one (it is a rather frequent case), while only two corners are needed to try and build a triangle, and so on.

The lower part of Fig. 6.20 shows the final result of the detection algorithm. Please notice that the algorithm is capable of detecting polygons also in a number of difficult (but not uncommon) cases:

- Polygons featuring appendices stemming from sides and/or corners (Fig. 6.20, third symbol from the left).
- Frags representing two different overlapping glyphs (Fig. 6.20, fourth symbol on the left).
- Polygons which have been partially drawn (Fig. 6.20, last symbol from the left)

#### 6.2.4 OGR area inference

The purpose of the second step of the recognition is to analyze each frag in order to infer its OGR area (see Section 5.3). The inference can be performed only for those frags which have been associated to a detected base shape. Otherwise, if no shape was detected for a given frag during the first step of the recognition algorithm, the system cannot perform any inference, since there is an insufficient amount of information to work with.

The key requirements to build a good inference algorithm are statistical analyses and a sufficient understanding of the most distinguishing features of each OGR area. Statistical analyses, performed in advance during a training phase, come mainly into play when evaluating the type and the size of a detected shape. In most cases, such evaluation is sufficient to carry out the inference of the OGR area. In practice, inference algorithms for different



Figure 6.20: Polygon detection algorithm implemented within the SW-OGR Engine, applied to glyphs featuring rectangular shapes. The algorithm uses the Shi and Tomasi method to locate the corners in each image (top). Afterwards, the corners are processed in order to build the polygon, if this is possible (bottom).

OGR areas perform different evaluations on the detected shapes. As an example, any glyph within the Shoulders Movements area is characterized by a filled rectangular shape with a very small height and a very large width, and the latter typically spans most of the width of the slice (see Fig. 6.21). As a consequence, to ascribe a glyph to Shoulders Movements area, it is appropriate to calculate the following:

- The ratio of width to height of the detected rectangle.
- The ratio of the width of detected rectangle to the width of the slice.
- The foreground filling of the rectangle (i.e. if it is filled or not).

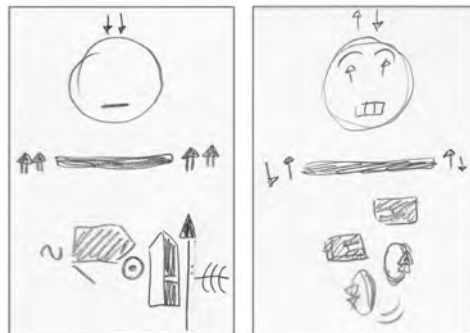


Figure 6.21: Two examples of signs containing glyphs belonging to the Shoulder Movements (ShM) area. The glyphs are visible right below the head circle.

In order to perform the evaluations in the above list, it is mandatory to perform a number of statistical analyses. More specifically, the SW-OGR Engine requires threshold values to check the features of the detected shape which is being analyzed. The analysis was performed on each text of the DS-PEAR and DS-POITIERS datasets. For each text, we measured different features

within glyphs belonging to different OGR areas. Such measurements are of critical importance during the OGR area inference step. Of course, not every evaluation needs a statistical background (e.g. checking the absence of a given type of detected shape within a frag, etc.).

To provide a detailed view of the analysis and development effort behind the classes which perform the inference of the OGR area, the following section reports one of the most representative cases: the Head Circles and Contacts area.

It is important to remind that if an OGR area inference procedure is successfully performed, a `Glyph` object is created. From that moment on, the SW-OGR Engine mainly works on glyphs (rather than frags).

#### 6.2.4.1 Head Circles and Contacts (HeE\_HE)

The inference of the Circle and Contacts area is performed by the `ModelHeE_HE` class (Section 6.1.2), and it is triggered if and only if a circle is detected within a given frag (Section 6.2.3.1). The `ModelHeE_HE` class performs the following checks over the frag:

- The ratio of the diameter of the circle to the width of the slice must exceed the `MIN_HEAD2SLICE_RATIO` constant.
- The frag must not be associated to any detected triangle, otherwise the frag most likely belongs to one of the movement areas. If a triangle is associated with the frag, it must not present any appendix stemming from the sides.
- (Optional) The frag should enclose other different frags (eyes symbols, mouth symbols, etc.). Such requirement is optional because it is possible to draw an empty head circle within a `SignWriting` sign, without specifying any facial expression.

It is evident that the only evaluation (among those reported in the above list) requiring a statistical background is the first one, i.e. the check on the ratio of the diameter of the circle to the width of the slice. It is worth noticing that we do not deal with thresholds related to absolute values, but to ratios between core elements, so that it is possible to process frags/glyphs of different size. In other words, we deal with rations among slice elements, instead of considering absolute measures. In the case at hand, analyzing each text within the DS-PEAR (58 pages) and DS-POITIERS (156 pages) datasets, i.e. 214 pages, we isolated each head circle, and calculated the ratio of its diameter to the width of its slice. The results of our analysis are reported in Tab. 6.5. It is evident that the diameter of most of the head circles covers about the 30% of the width of the slice. This can also be empirically verified by observing the texts within our datasets. In order to make the OGR area inference more resilient (many head circles do not reach the 30% threshold), we chose to use the  $\mu - \sigma$  formula (average minus standard deviation) to set the lower bound of the ratio, so the numeric

constant `MIN_HEAD2SLICE_RATIO` is equal to 0.2137924 (see also Fig. 6.8). No upper bound for the ratio has been set so far, since the head circles are in any case the largest ones, within a `SignWriting` text.

#### Ratio of the diameter of head circles to the width of their slices

Avg. ( $\mu$ )	Std.Dev. ( $\sigma$ )	Min	Max
0.3067719	0.0929795	0.0978474	0.6145251

*Table 6.5:* Ratio of the diameter of head circles to the width of their slices.

The inference for the Head Circles and Contacts area is a very unique procedure within the whole SW-OGR Engine, since it presents one important exception. In fact, the head circles are the only symbols, within the entire ISWA, which may contain other symbols (eyes, mouth, etc.) inside, in a structured way. More specifically, `SignWriting` basically allows the users to draw any symbol within (or upon) any other symbol, if it is possible, and if it makes sense from a linguistic and production point of view. The head circles, for obvious reasons, impose a structured disposition on the symbols they contain. In fact, it is “very unlikely”, to see a mouth symbol in the place where a forehead symbol should be, etc.

Therefore, identifying a head circle makes a great deal of information available about any possible frag it may contain. For this reason, once that a frag has been successfully inferred to belong to the Head Circles and Contacts area, it is analyzed to check if it contains any frag. For each frag found in this way, the SW-OGR Engine infers its OGR area (eyes, mouth, etc.) by performing two assessments:

- The position of the head circle center with respect to the Minimum Bounding Rectangle MBR of the contained frag (see Fig. 6.22). For this, we exploited the classification of topological relationships between points and MBRs defined in (Papadias et al., 1995).
- The relationship between the MBR of the head circle and the MBR of the contained frag (see Fig. 6.23)

The purpose of the first check aims at defining the most likely OGR area for the glyph. The position of the head circle center with respect to the MBR of the frag (Papadias et al., 1995) is a valuable information, but it is often not enough to complete the inference. In fact, as an example, both eyes and forehead symbols can be placed north of the head circle center. Fig. 6.22 illustrates the first check, applied to frags representing left eyes. The left side of the figure shows the detected circle within the frag, the green highlighted area within the circle identifies the region in which a frag representing a left eye is allowed to occur. This concept is further elaborated in the right part of the figure: all the possible relationship between the center of the detected circle and the MBR of a frag representing a left eye are highlighted. Please

notice that each MBR has a different topological relationship with the center of the circle.

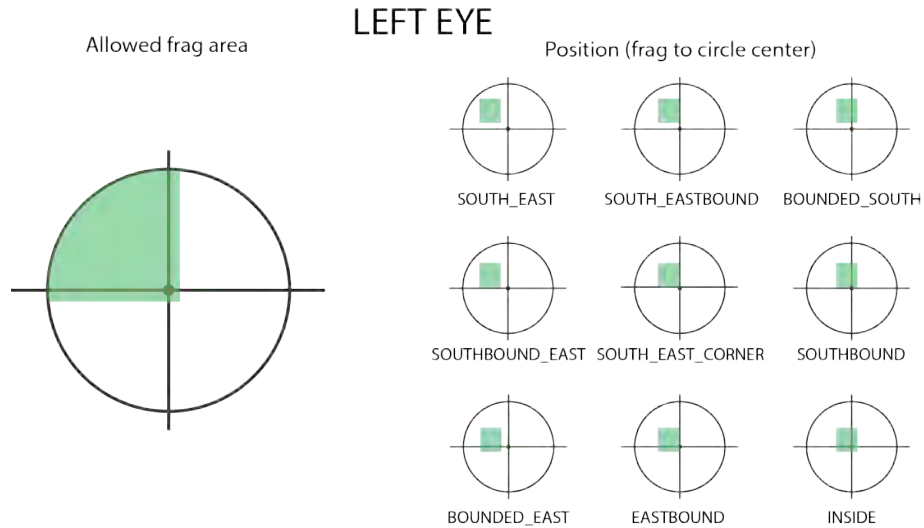


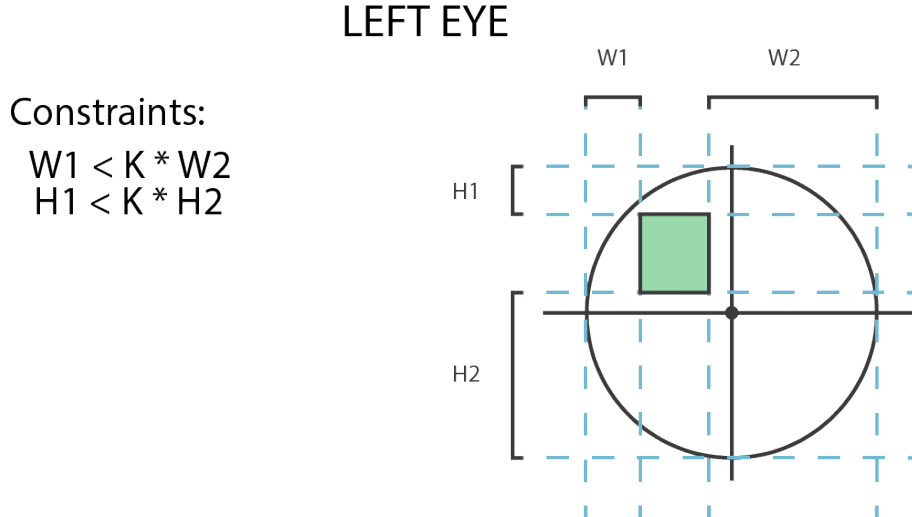
Figure 6.22: OGR area inference check for facial expressions. A frag is inferred to represent a left eye according to the topological relationship between its MBR and the center of the head circle.

The purpose of the second check is to confirm (or correct) the outcome of the first check by evaluating the relationship between the MBR of the head circle and the MBR of the frag. The second check is typically carried out through the comparison of the coordinates of the MBRs, and evaluating their distances. Fig. 6.23 shows the second check applied to frags representing left eyes. Observing the figure, it is possible to notice that the check is performed both on the x-axis and y-axis. More specifically, the check is performed by calculating:

- W1: the x-axis distance between the top-left point of the MBR of the detected circle and the top-left point of MBR of the frag.
- W2: the x-axis distance between the top-right point of the MBR of the detected circle and the top-right point of MBR of the frag.
- H1: the y-axis distance between the top-left point of the MBR of the detected circle and the top-left point of MBR of the frag.
- H2: the y-axis distance between the bottom-left point of the MBR of the detected circle and the bottom-left point of MBR of the frag.

To perform the inference of the OGR area with success, the following rules must hold:  $W1 < K * W2$ , and  $H1 < K * H2$ , where  $K$  equals to  $1/3$ . In other words, in order to successfully perform the inference, the MBR of the

frag must be very close to the top-left point of the MBR of the detected circle.



*Figure 6.23:* OGR area inference check for facial expressions. A frag is inferred to represent a left eye according to the topological relationship between its MBR and the MBR of the head circle.

#### 6.2.4.2 Hand Configurations (HaC)

In spite of the complexity of the Hand Configurations, evaluating the features of a glyph and inferring its membership to such OGR area is trivial. The base shapes and a few other features, in fact, are sufficient to recognize if a glyph belongs to the Hand Configurations area. In fact, most hand configurations glyphs present very particular base shapes which are not present into any other OGR area. In Chapter 5 (Tab. 5.3 in particular), we presented the full list of the base shapes of the glyph belonging to the Hand Configurations area. As a remainder, such shapes have been identified as:

- SQUARE
- RECTANGLE
- CIRCLE
- HOUSE
- R-HOUSE
- TRAPEZIUM
- SPIRAL

As anticipated, a number of such base shapes can be found only within hand configurations symbols, such shapes are SQUARE, HOUSE, R-HOUSE,



TRAPEZIUM. On the opposite, RECTANGLE and CIRCLE are found in other OGR areas, so they can be assigned to the Hand Configurations area only by evaluating their dimensions. More specifically, RECTANGLE shapes undergo a check of the height-to-width ratio, while CIRCLE shapes are inferred by checking the ratio of the diameter to the width of the slice. Finally, the SPIRAL shape can be found in other OGR areas. Only in the case of the Hand Configurations, however, the SPIRAL shape has appendices stemming from it. As a consequence, the neighborhood of the glyphs featuring a SPIRAL shape is analyzed to look for possible appendices, if any is found, then the glyph belongs to the Hand Configurations area.

### 6.2.5 Glyph recognition

After the OGR area inference step, the SW-OGR Engine is ready to take on the final recognition step. During such step, the procedure analyzes each glyph and attempts to generate its OGR identifier (and, as a consequence, its ISWA identifier), exploiting any available information gathered during the previous steps.

Even though the classes devoted to the final recognition have the same structure (Section 6.9), their business logic can be very different. Consistently with the organization of the chapter, it is worth showing the implementation of two recognition classes with different levels of complexity. Section 6.2.5.1 describes `RecogHeE_HE`: the recognition class for the glyph belonging to the Head Circles and Contacts area. Section 6.2.5.2 describes the most challenging class within the SW-OGR project, i.e. `RecogHaC`: the recognition class for the glyph belonging to the Hand Configurations area.

#### 6.2.5.1 Head Circles and Contacts (`HeE_HE`)

The Head Circles and Contacts area contains a very restricted number of glyphs. The glyphs are very similar among themselves, since they all represent head circles, and we are not dealing in this area with what can be included inside the circles, i.e., eyes, nose and mouth. The glyphs differ by a few, localized details, which can be found in different regions. Fig 6.24 shows the most representative glyphs within such area. Please notice the different features along the circle representing the head.

We decided to exploit our knowledge of the available features of the glyphs



Figure 6.24: A set of glyphs belonging to the Head Circle and Contacts area.

of such area, and their restricted number, by designing a number of *ad hoc* evaluations. More specifically, given a glyph representing a head circle, the

recognition algorithm inspects the image in a number of regions that may contain distinctive features. In order to do that, we took advantage of two very basic image processing tools:

- The computation of the convex hull containing the glyph (with convexity defects evaluation).
- The run-length encoding (Lynch, 1985).

As an example, detecting the presence of a neck attached to the head circle is performed by detecting the presence of major convexity defects in the bottom part of the image. The presence of foreground-filled features (rightmost symbol in Fig. 6.24) is detected by combining the evaluation of the convex hull with that of the run-length encoding of the image. Fig 6.25 shows 3 screenshots, taken during the test phase of SW-OGR. Each glyph within the figure shows a different feature, the colored square around each feature shows that the recognition algorithm has successfully detected the features. Once the features are detected, the aggregation methods of the class generate the OGR code for the glyph, which in turn allows retrieving its ISWA code. The detected codes are stored, in order for them to be retrieved when producing the output data of the recognition.



Figure 6.25: Feature recognition applied to a set of glyphs belonging to the Head Circle and Contacts area.

### 6.2.5.2 Hand Configurations (HaC)

In order to identify the OGR code of a glyph belonging to the Hand Configurations area, a high amount of information is required. Full details about the structure of the OGR coding is reported in Section 5.3.2. It is important to remind, however, that each hand configuration symbol is composed by a base shape, which represents the hand. Such shape is typically a rectangle, but circles, trapeziums and other shapes are also common. The fingers of the hand are represented by a number of segments (appendices) placed near (or attached to) the base shape. Fig. 6.26 shows a number of hand configurations taken from the DS-PEAR dataset, along with their detected base shapes.

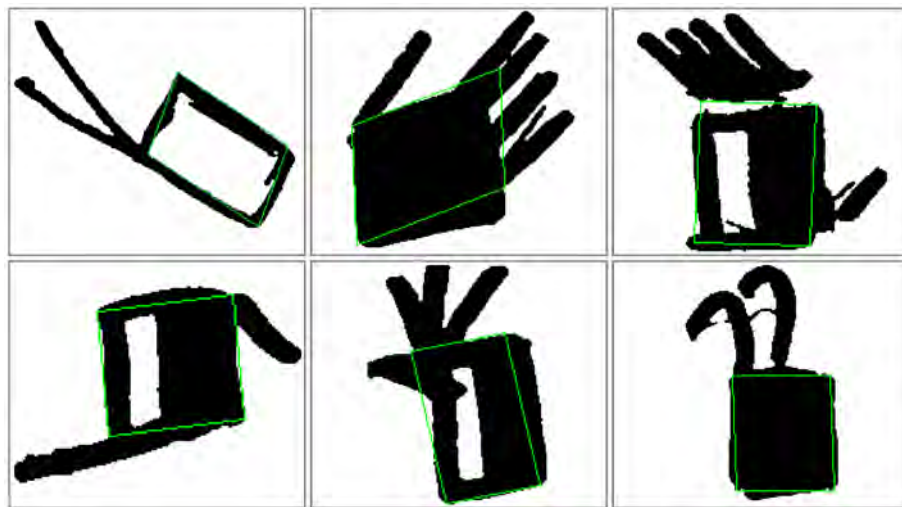
The key image features of each hand configuration symbol can be divided into two sets:

- Features related to the base shape, such as type and color filling of the shape.
- Features related to the appendices, such as number, shape, angle, contacts with other appendices, etc.

The first set of features is trivial to evaluate: the type of the shape (rectangle, circle, etc.) has already been identified during the first recognition step, and the information is available to the recognition class. The color filling can be easily evaluated inspecting the pixels within the shape, since its bounds have already been identified too.

The most challenging set of features is actually the second one. First of all, the appendices, being handwritten, and due to digitalization problems, often show as quite thick clusters of foreground pixels whose features, e.g. shape, direction or crossing, can be difficult to evaluate. For this reason, we decided to apply the thinning algorithm of Zhang and Suen (T. Zhang & Suen, 1984).

Thinning the image reduces any line within the image to a 1-pixel-wide



*Figure 6.26:* A set of glyphs belonging to the Hands Configurations area. The detected base shape is marked over each glyph.

line. This proves very useful when evaluating the features of the appendices, since it makes detecting their shape, direction, etc. much more easier. Of course, this may have a serious impact over the integrity of the base shapes. As you can see, in a number of glyphs in Fig. 6.27, a foreground filled base shape is reduced to a 1-pixel wide line. However, this causes no problem, since the evaluation of the base shape and the evaluation of the appendices are carried out separately. Fig. 6.27 shows the symbols already introduced in Fig. 6.26, as they appear after the thinning process.

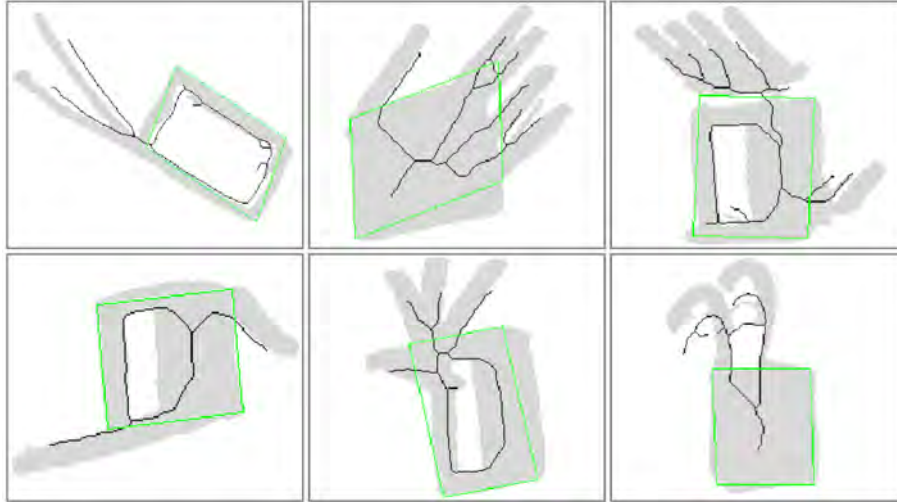


Figure 6.27: A set of glyphs belonging to the Hands Configurations area, as they appear after the thinning process. The detected base shape is marked over each glyph, and the original foreground area is visible in transparency behind each glyph.

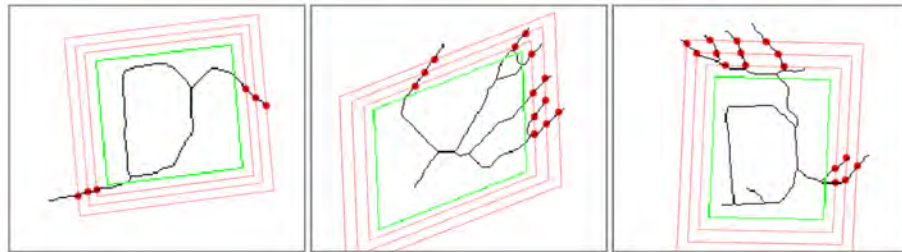
We decided to begin the detection of the appendices after the thinning operations, by locating their *start points*. In this context, we defined a start point to be a point:

- which is part of the base shape (attached appendices) or as close as possible to it (detached appendices),
- from which the appendix stems.

In order to locate the start points, we exploited our knowledge about the base shape detected within the glyph. In fact, the algorithm looks for foreground pixels in the outer neighborhood of the base shape. The start points of the inner appendices, on the other hand, are located by scanning the inner neighborhood of the base shape. The foreground pixels detected using such method will most likely belong to an appendix.

The neighborhood check is performed by generating concentric shapes through scaling the base shape detected within the glyph by a 0.10 factor for a number of times. Of course, the shape is enlarged to look for outer appendices, and shrunk for inner appendices. The perimeter of each scaled shape is referred to as *scan line*. Any foreground pixel located along the generated scan lines is considered as a possible *start point* of an appendix. Fig. 6.28 illustrates the start point detection of the outer appendices, applied to a number of glyphs already introduced in Fig. 6.26. A number of scan lines are visible on the outer neighborhood of the detected base shape of each glyph, and the dots represent the detected foreground pixels along the scan lines. Notice that, as the description of the algorithm goes on, the number of details to be shown for each glyph increases, so we will be using a reduced number of glyphs as examples, chosen among the most representative cases.

By observing Fig. 6.28, it is possible to notice that each appendix is often detected multiple times through the different scan lines it intersects. To address this issue, after the inner and outer scans, the SW-OGR Engine optimizes the detected points. Points which belong to the same appendix (i.e. very close and connected among them) are merged into one single point, namely the one closest to the base shape is chosen, since it represents the most reasonable starting point of the appendix. This optimization is performed for two reasons: the first one is, of course, to reduce the number of points to work with. The second one is to make sure that the detected points are as close as possible to the base shape, in order for them to be identified with the actual start point of the appendix.



*Figure 6.28:* Start point detection of the outer appendices, applied to a number of glyphs belonging to the Hands Configurations area. The detected base shape is marked over each glyph, the scan lines are visible around the base shape, and the detected foreground points along the scan lines are marked by dots.

After the detection of the start points of the possible appendices, the algorithm “follows” the appendices to detect the full set of foreground lines stemming from each start point. Since handwriting may often be very inaccurate, and since the thinning procedure may sometimes introduce short spurs in spite of any pruning procedure (Lam, Lee, & Suen, 1992), the number of foreground lines stemming from one single point can be surprisingly high. However, it is also very common to have one single line stemming from one single start point (as it should be in most cases). The foreground lines are analyzed in a specific direction: outward with respect to the base shape, if following an outer appendix, and inward if following an inner appendix. Adopting a very consolidated methodology (Sarfranz, 2005) in the field of computer vision, during the analysis, the following points are marked, and we will refer to them using the term Point of Interest (POI)

- *End points:* foreground pixels featuring only one neighbor (Fig. 6.29).
- *Branch points:* foreground pixels featuring three neighbors (Fig. 6.30).
- *Cross points:* foreground pixels featuring four neighbors (Fig. 6.31).

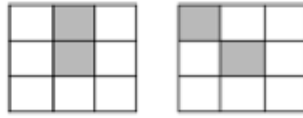


Figure 6.29: Examples of end points in a line.

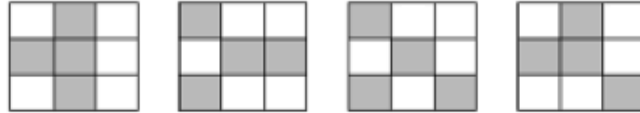


Figure 6.30: Examples of branch points in a line.

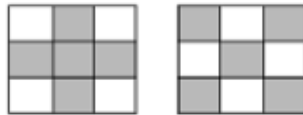


Figure 6.31: Examples of cross points in a line.

In this initial stage of the detection, if a foreground line connects two POIs, it is treated as an independent appendix. As a consequence, it is not uncommon to find glyphs which carry up to 30 initial appendices. Fig. 6.32 shows the initial phase of the outer appendix scan, applied to the a number of glyphs shown in Fig. 6.26. Each column of the image shows:

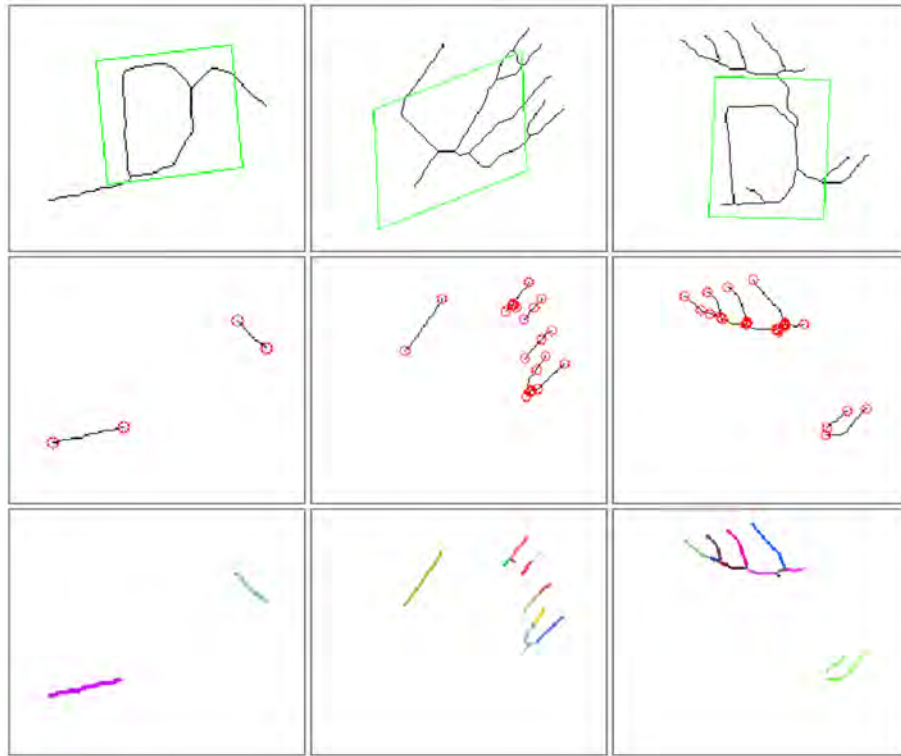
- The image of the glyph, along with its detected base shape.
- End, branch and cross points (POIs) detected following the appendices.
- The detected appendices, each one highlighted with a random color.

Please notice that, in order to avoid false positives while following the appendices, during the scan for the outer appendices, the region occupied by the base shape is erased, so that the appendix scan may proceed only on the outer vicinity of the glyph. The opposite happens when following the inner appendices: anything outside the detected base shape is erased.

As anticipated, the initial set of appendices can be quite large, often in the order of tens. As a consequence, we devised an optimization procedure, composed by different checks, in order to decrease the cardinality of the set to a maximum value of 5. Such checks range from very simple ones, such as a check on the minimum length of an appendix, to more complex ones. The checks are all described in the following. They are iterated in the order listed below until the set of appendices cannot be further reduced (i.e. no check can be applied anymore). The first optimization check is a security one. It basically checks for the existence of POIs which delimit no appendix. This

does not occur during the initial iterations, but it may happen as a result of the other optimization checks, which sometimes perform the removal of an appendix. If a POIs with 0 appendices is found, it is erased.

The second optimization check is performed on appendices, instead of POIs.



*Figure 6.32:* Illustration of the initial outer appendix scan applied to a number of glyphs belonging to the Hands Configurations area. Each column of the image shows: the image of the glyph, along with its detected base shape; the POIs detected following the appendices; the detected appendices, each one highlighted with a random color.

If the length of the appendix is smaller than a threshold value  $T$ , the appendix is erased.  $T$  is calculated by multiplying a numeric constant  $K$  to the average length of the sides of the base shape.

The third optimization check is performed if a POIs  $P$  delimits two appendices, say  $A$  and  $B$ . Again, this situation can occur after the removal of an appendix, performed during an optimization check, or if two appendices share the same start point. If the direction of the appendices is compatible (i.e.  $A$  heads North, and  $B$  North-East, or they both head North), the appendices are merged, and  $P$  is erased (see Fig. 6.33). If  $A$  and  $B$  are merged, the new resulting appendix will have the start point of  $A$  and the end point of  $B$ , and it will carry the points of both appendices. If the appendices are not merged, it will most likely mean that the point is a start point, from which two different appendices stem. This information is very useful when

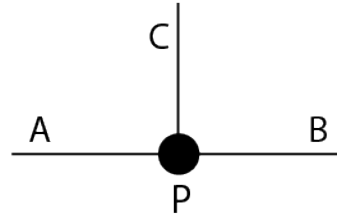
assessing contacts and intersections between appendices. As a consequence, it is saved onto an appropriate data structure.

The fourth optimization check is oriented to the branch points. This is the

*If*

Point  $P$  has 2 appendices:  $A$  and  $B$ .

The directions of  $A$  and  $B$  are compatible.



*Then*

Merge  $A$  and  $B$ .

Erase  $P$ .

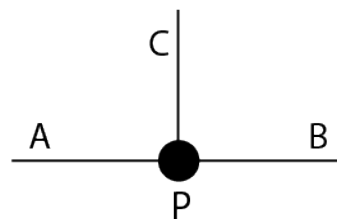
*Figure 6.33:* Illustration of the third appendix optimization step, performed during the recognition of glyphs belonging to the Hands configuration area.

case when a POIs  $P$  delimits three appendices, say  $A$  and  $B$  and  $C$ . If the directions of two of such appendices, say  $A$  and  $B$ , are compatible, the appendices are merged (see Fig. 6.34). If the length of  $C$  does not exceed a threshold value  $T$ , then  $C$  is erased and, as a consequence,  $P$  is erased (since it does not delimit any appendix anymore). If all three appendices have compatible directions, the pair to be merged is composed by the longest ones.

*If*

Point  $P$  has 3 appendices:  $A$ ,  $B$  and  $C$ .

The directions of  $A$  and  $B$  are compatible.



*Then*

Merge  $A$  and  $B$ .

If the length of  $C$  does not exceed a threshold value  $T$

Erase  $C$ .

Erase  $P$ .

*Figure 6.34:* Illustration of the fourth appendix optimization step, performed during the recognition of glyphs belonging to the Hands configuration area.

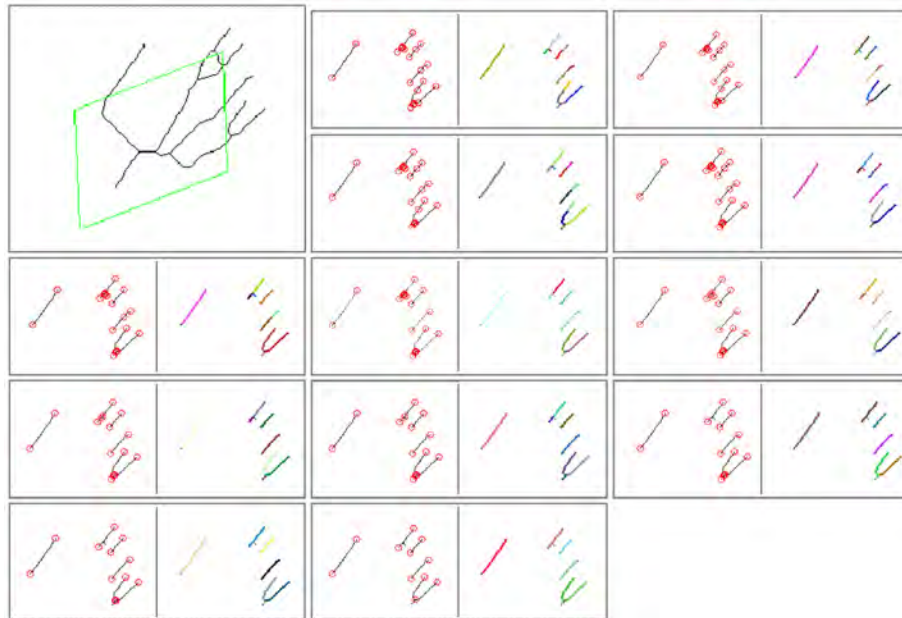
Finally, the last optimization check deals with cross points, i.e. POIs with 4 appendices. During this optimization step, the pairs of appendices with compatible directions are merged, and the cross point is erased. The contact



between the two new appendices are saved onto the proper data structure, in order to be used later.

We tested our algorithm on a wide range of glyphs, picked from every dataset we used. We observed that the optimization algorithm is very effective at decreasing the number of detected appendices. Fig. 6.35 and Fig. 6.36 show the algorithm in action on the glyphs introduced in Fig. 6.32 (except the first one, since it does not need any optimization). It is possible to notice that, step by step, the number of POIs and appendices of the glyphs is progressively reduced. Please notice that sometimes the differences between two successive snapshots in are very hard to spot, since the POIs can be very dense.

The data structures of the `RecogHaC` class are designed to store any data



*Figure 6.35:* Illustration of the fourth appendix optimization step, performed during the recognition of glyphs belonging to the Hands configuration area.

gathered during the optimization steps. More specifically, it is of critical importance to store at least the following information:

- The points which compose the appendix, from the start point, to the end point.
- The contacts (if any) of each appendix with the other appendices.

By performing different evaluations over the points of the appendix, the algorithm can extract most of the information needed to build the OGR code of the glyph (Section 5.3.2). More specifically, the points of an appendix can be used to identify its shape, closest side, angle with the closest side, etc. The information about the contacts between appendices comes also into play when coding each appendix into the OGR code of the glyph.

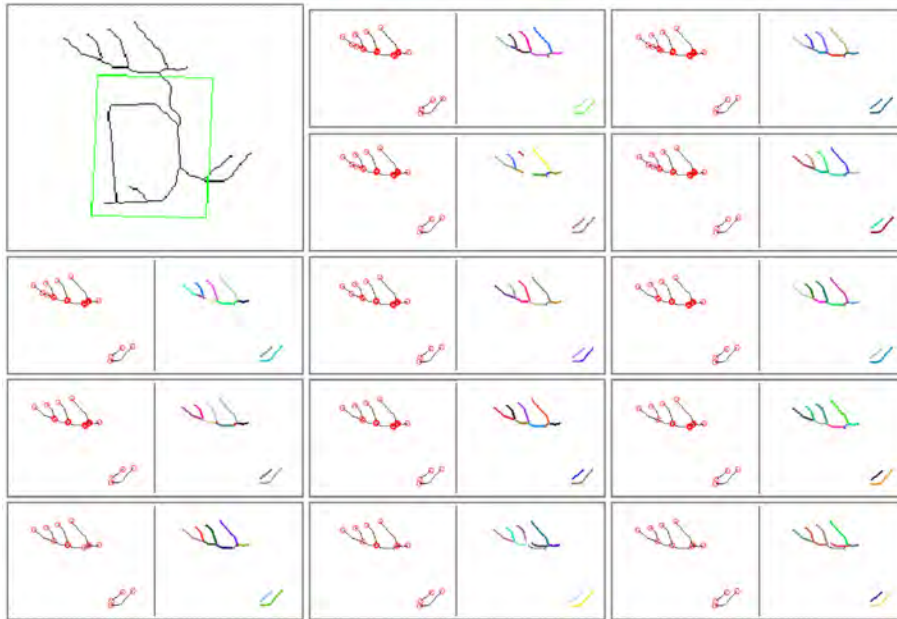


Figure 6.36: Illustration of the fourth appendix optimization step, performed during the recognition of glyphs belonging to the Hands configuration area.

As explained in Section 6.1.2, any information detected about the glyph is gathered by the aggregation methods of the recognition class *RecogHaC*. Such methods combine the data about the appendices with those about the base shape, in order to build the final OGR code of the glyph.

Once the OGR code is created, the finalization methods use it as a key to access the mapping tables of the application. The mapping tables (introduced in Section 5.4) are simple text files that match OGR codes with ISWA codes. As a result, they allow the system to switch from an application-specific coding system (OGR) to a worldwide-accepted coding system (ISWA). Sometimes, the OGR code is not 100% accurate, and it cannot be found “as is” within the mapping tables. This may be due to different reasons, including inaccurate handwriting or general detection issues. For this reason, the finalization methods do not look for an exact match for the OGR code within the mapping table. They rather evaluate a similarity score based on the Levenshtein distance<sup>3</sup> between OGR codes, choosing the most similar one. Once the glyph has been identified, its ISWA and OGR codes are stored, in order to be accessed at the end of the recognition procedure, when producing the output data structures.

Fig. 6.37 and Fig. 6.38 summarize the recognition of two of the example hand configuration symbols that have been used so far. Within each pic-

<sup>3</sup>In information theory, the Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other.

ture, please notice: the original handwritten symbol (top-left), the detected points and appendices (top-center and right), the recognized glyph (bottom) and finally, written on the last line of the console (bottom) the ISWA code of the glyph.



Figure 6.37: Screenshot representing the completed recognition of a glyph belonging to the Hands Configurations area.

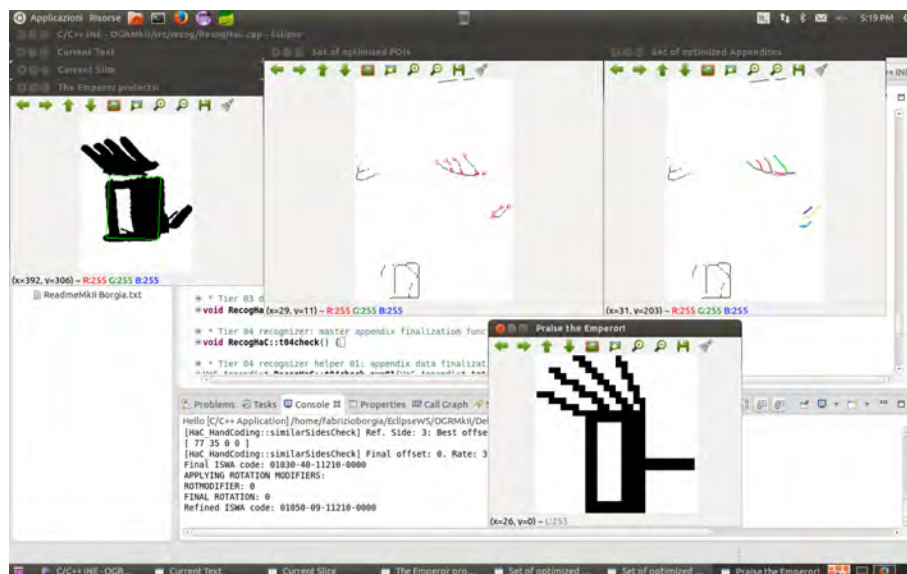


Figure 6.38: Screenshot representing the completed recognition of a glyph belonging to the Hands Configurations area.

A very similar recognition algorithm has been employed for other OGR areas, such as the Hand Movements area. In fact, the corresponding algo-

rithm is based on the recognition of a base shape (strictly triangles), and the evaluation of its appendices, which, unlike the hands configurations, do not represent fingers, but the lines of an arrow.

### 6.3 Development status and limitations

Despite the design of the SW-OGR application is concluded, at least in its core lines, its development is still ongoing. Developing and testing the business logic necessary to support each ISWA area is a very time-consuming task. Despite this, the application framework is ready, and the recognition works for the presently supported OGR areas. In fact, the SW-OGR Engine is a modular application: as soon as the work for an OGR area is completed, it is included into the main branch of the project.

Tab. 6.6 summarizes the development status for each OGR area, updated to the 27th of August 2014.

**Development status of the SW-OGR Engine**

Code	Description	Status
HaC	Hand Configurations	80% Completed. Further testing required.
HaM	Hand Movements	80% Completed To be tested.
ArP	Arm Positions	Designed.
ArM	Arm Movements	Designed.
ShP	Shoulders Positions	40% Completed.
ShM	Shoulder Movements	40% Completed.
HeE_HE	Head Circles Head Contacts	Completed. Tested.
HeE_EY	Facial Expressions: Eyes	Completed. Tested.
HeE_NO	Facial Expressions: Nose	Completed. Tested.
HeE_MO	Facial Expressions: Mouth	80% Completed. Tested.

*Continued on next page*

**Development status of the SW-OGR Engine** – *Continued from  
previous page*

Code	Description	Status
HeM	Head Movements	Designed.
Co	Hand Contacts	Designed.
Dy	Dynamics	Designed.
Cr	Coordination	Designed.
Pu	Punctuation	80% Completed To be tested.

*Table 6.6:* Development Status, updated to the 27th of August 2014.

Furthermore, it is important to mention one limitation of our SW-OGR Engine. We observed that sometimes, SignWriting texts may contain overlapping glyphs, i.e. glyphs which have been drawn one upon the other. In those cases, distinguishing one glyph from another can be very tricky. As explained in Chapter 4, the SW-OGR Engine is able to perform both online and offline recognition. In the first case (online), the system recognizes the text as it is being written. As a consequence, temporal information about the composition is available, so the problem of overlapping glyphs is extremely rare, since the user typically draws one symbol (or a part of a symbol) at a time. In the second case (offline), on the other hand, no temporal information is available and it is very difficult (sometimes also for a human being) to distinguish all elements of one glyph from those of another one. For this reason, the SW-OGR Engine currently does not recognize overlapping glyphs. As shown in Section 6.2.3, the system is already able to identify a shape, even if there are two overlapping shapes, that could be a good point to start. A further design and development effort is necessary, however, to overcome such limitation.



## Chapter 7

# Testing of SW-OGR

### 7.1 Testing of SW-OGR

As mentioned in the previous section, the application has been developed in a modular way: as soon as the code supporting a particular OGR area is ready, it is included within the main project. For this reason, however, a global testing of the whole SW-OGR Engine has not yet been performed. More specifically, only a qualitative analysis of the execution of the global recognition algorithm is available.

On the other hand, before including code supporting any OGR area within the project, the code is thoroughly tested against the datasets we use. As shown in Tab. 6.6, we decided to focus our attention of the most representative OGR areas first, i.e. the areas which are most frequently used within a SignWriting text. Therefore, quantitative analyses are available for the following areas: Head Circle and Contacts, Hand Configurations and Facial Expressions: Eyes.

The dataset gathered for the tests has been assembled by extracting 20 pages both from the DS-PEAR and the DS-POITIERS dataset: we will be referring to this dataset as *DS-TEST*. Any code included into the project has been tested at least against DS-TEST. Some OGR areas, anyway, required a larger amount of testing, especially the most complex ones, such as Hand Configurations. In such cases, we also evaluated the behavior of our code by using the DS-PRINTED dataset, and we generated (and tested the code on) images of the DS-DEV datasets.

After gathering the dataset, it was necessary to generate a ground truth file for each image, in order to enumerate the glyphs contained within the page. More specifically, the ground truth file is an XML file, generated by humans, which holds the code and the coordinates of any glyph within the text. In other words, it is the product of a human-performed SW-OGR, which is used to assess the reliability of the automatic recognition.

The quantitative analyses are performed separately for each dataset. Moreover, since the DS-TEST dataset contains pages extracted from two different datasets (DS-PEAR and DS-POITIERS), it has been analyzed separately.

As a results, in each table of the present section, we will be using the label *DS-TEST (POITIERS)* when presenting the analysis of the performance of the SW-OGR with the pages of DS-TEST which have been extracted from DS-POITIERS, and we will use *DS-TEST (PEAR)* for the other half of the dataset.

Four categories have been identified to evaluate the performance of the SW-OGR Engine:

- **Recognized:** if a glyph has been correctly recognized.
- **Inaccurate:** if a glyph has been correctly inferred to belong to the OGR area which is being tested, but its recognition resulted in a wrong ISWA code.
- **Not recognized:** if a glyph belongs to the OGR area which is being tested, but the SW-OGR Engine was not able to perform the OGR area inference, thus resulting in no recognition at all.
- **False positive:** if a glyph does not belong to the OGR area which is being tested, but the SW-OGR Engine erroneously included it into the recognition, thus resulting in a false positive.

The percentages (calculated over the whole dataset, not over the single pages) of such categories for each OGR area are reported into each table of the present section.

### 7.1.1 Head Circles and Contacts (HeE\_HE)

Given its low level of complexity, the code supporting the Head Circle and Contacts area was the first one to be included within the test, in chronological order. The test has been performed by taking into account the DS-TEST only.

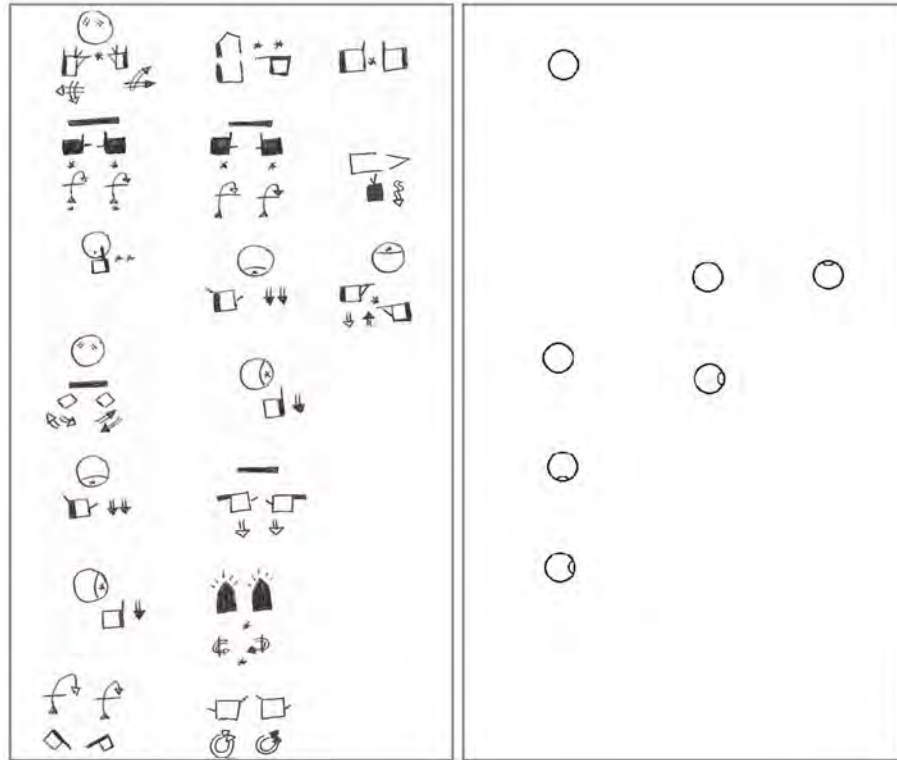
#### SW-OGR performance test: Head Circle and Contacts

Recognition	DS-TEST-PEAR	DS-TEST-POITIERS
<b>Recognized</b>	94.2%	97.1%
<b>Inaccurate</b>	2.8%	1.8%
<b>Not recognized</b>	2.8%	1.1%
<b>False positives</b>	0.2%	0%

Table 7.1: SW-OGR performance test: Head Circle and Contacts.



The test results for this OGR area are very encouraging, and they are reported in Tab 7.1. The results for the Head Circle and Contacts area are quite satisfying: the average of the performance score being 95.6%.



*Figure 7.1:* A handwritten page belonging to the DS-TEST-POITIERS dataset, as it appears before (left) and after (right) the recognition of the glyphs belonging to the Head Circles and Contacts area.

We observed that most of the errors in the recognition were caused by extremely inaccurate handwriting (1.2% in DS-TEST-PEAR and 0.8% in DS-TEST-POITIERS), and overlapping glyphs (1.6% in DS-TEST-PEAR and 1.0% in DS-TEST-POITIERS).

We also detected a small percentage of unrecognized glyphs within each dataset. One of the reason for this failure in the recognition is the presence of overlapping glyphs (1.5% in DS-TEST-PEAR and 0.5% in DS-TEST-POITIERS). Anyway, within both datasets, a significant percentage of unrecognized glyph (1.3% in DS-TEST-PEAR and 0.6% in DS-TEST-POITIERS) is justified by the fact that some pages within such datasets feature large slices with very tiny head circles. As a consequence, the diameter of the circles is smaller than the minimum threshold value necessary to be included within the Head Circle and Contacts area.

Finally, the small percentage (0.2%) of false positives within the DS-TEST (PEAR) has been observed to be due to an improper OGR area inference

of a very restricted number of circular movements.

Fig. 7.1 shows a handwritten page belonging to the DS-TEST-POITIERS dataset, as it appears before (left) and after (right) the recognition. We decided to include this particular screenshot since it shows one of the issues mentioned above, i.e. the failure of the recognition due to the presence of overlapping glyph. In fact one of the head circle in the first column is overlapped with a hand configuration glyph, and therefore the recognition for that particular symbol has failed (it is not shown in the output image on the right side of Fig 7.1).

### 7.1.2 Facial Expressions: Eyes (HeE\_EY)

The support for Facial Expressions: Eyes area is the latest, in chronological order, to be included within the SW-OGR Engine. The number of glyph within this area is not particularly small, but the symbols have the advantage of being very simple. As a consequence, they are quite easy to draw (with an acceptable level of accuracy), and just as easy to recognize. Since no glyph belonging to the Facial Expressions: Eyes area has been dealt with in this Chapter, Fig. 7.2 shows a small selection of such symbols. Please notice that the head circles are not necessarily part of the glyphs, more specifically, each glyph in this area comes in two ways: with and without the head circle.



*Figure 7.2:* A set of glyph belonging to the Facial Expressions: Eyes area.

The results of the test for the Facial Expressions: Eyes area are shown in Tab. 7.2. The recognition procedure has been tested using the DS-TEST dataset, and its performances were very encouraging, since it reached an average value of 94.9.

The total lack of false positives, and the very low percentages of recognition inaccuracies (1.2% for DS-PEAR and 0.7% for DS-POITIERS) testify the robustness and the efficiency of the recognition algorithms.

It was interesting, however, to understand the reason for the remarkable percentages (5.1% for DS-PEAR and 3.2% for DS-POITIERS) of recognition failures. We tracked the reason for such failures in the presence of a number of eye symbols that have been drawn so close to their head circles to be included in the same frag, since they are connected. As a consequence, the OGR area algorithm ascribed those frags to the Head Circles and Contacts area, preventing the recognition of the eye symbol. This is just another issue deriving from the presence of overlapping glyphs within the SignWriting texts. A minor percentage of the recognition failures are also due to the presence of glyphs which are not yet supported by the recognition algorithm,

such as eye movements, which appear as small arrows.



Figure 7.3: A handwritten page belonging to the DS-TEST-POITIERS dataset, as it appears before (left) and after (right) the recognition of the glyphs belonging to the Facial Expression: Eyes area.

#### SW-OGR performance test: Facial Expressions: Eyes

Recognition	DS-TEST-PEAR	DS-TEST-POITIERS
Recognized	93.7%	96.1%
Inaccurate	1.2%	0.7%
Not recognized	5.1%	3.2%
False positives	0%	0%,

Table 7.2: SW-OGR performance test: Facial Expressions: Eyes.

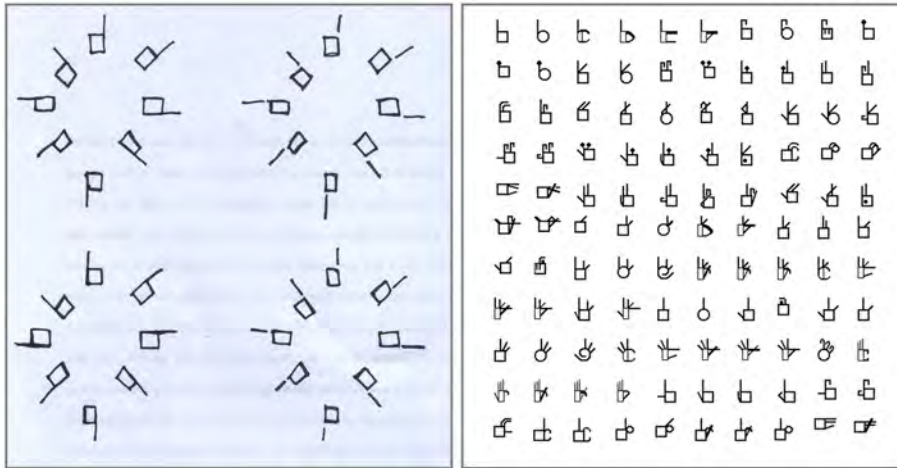
Fig. 7.3 shows a handwritten page belonging to the DS-TEST-POITIERS dataset, as it appears before (left) and after (right) the recognition. Please notice the two recognition failures within the column containing symbols

containing eye movements, which are not yet supported.

### 7.1.3 Hand Configurations (HaC)

RecogHaC, i.e. the recognition class for the Hand Configurations area, is a very complex one. As explained in Section 6.2.5.2, it features a huge set of interleaving evaluations. For this reason, during the development phase of the class, it was very difficult to perform debugging and testing with the handwritten images from DS-PEAR and DS-POITIERS, since their handwriting style is often very inaccurate.

As a consequence, in order to carry out very specific evaluations, we gen-



*Figure 7.4:* Two datasets generated to test the recognition algorithms for the Hand Configurations area. The page on the left represents a number of possible variations of a hand configuration. The page on the right represents a number of printed hand configurations.

erated a number of pages, including them within the DS-PRINTED or the DS-DEV dataset. For the DS-PRINTED dataset, we generated a number of pages containing printed hand configurations glyph, mainly to check the detection of the appendices (Fig. 7.4, right side). For the DS-DEV dataset, we generated a number of pages containing different variations for the same glyph, in order to evaluate our algorithms for the recognition of rotation, orientation, geometric plane and appendix position (Fig. 7.4, left side). Such datasets were used to test and tune the SW-OGR Engine during the early development phase. The tests were conducted in an informal way during the development of the application, thus no data is available. Once the SW-OGR Engine started to perform flawlessly with DS-DEV and DS-PRINT, we ran formal tests with the DS-TEST dataset. The results of the test are available in Tab. 7.3.

As shown in Tab. 7.3, the average success percentage on the DS-TEST is 68.25. As already shown in Section 7.1.1, the recognition algorithm performs

better with the DS-POITIERS dataset.

### SW-OGR performance test: Hand Configurations

Recognition	DS-TEST-PEAR	DS-TEST-POITIERS
<b>Recognized</b>	65.3%	71.2%
<b>Inaccurate</b>	19.3%	15.6%
<b>Not recognized</b>	15.3%	13.2%
<b>False positives</b>	0%	0%,

Table 7.3: SW-OGR performance test: Hand Configurations.

It is possible to detect a relevant percentage of recognition inaccuracies (19.3% for DS-TEST-PEAR and 15.6% for DS-TEST-POITIERS). This is due to a combination of a very inaccurate handwriting, the presence of glyphs which do not belong to the ISWA (Bianchini & Borgia, 2012; Bianchini et al., 2011) and a too strict feature evaluation routine. We think that this issue can be solved by further tuning the feature evaluation routine of the `RecogHaC` class.

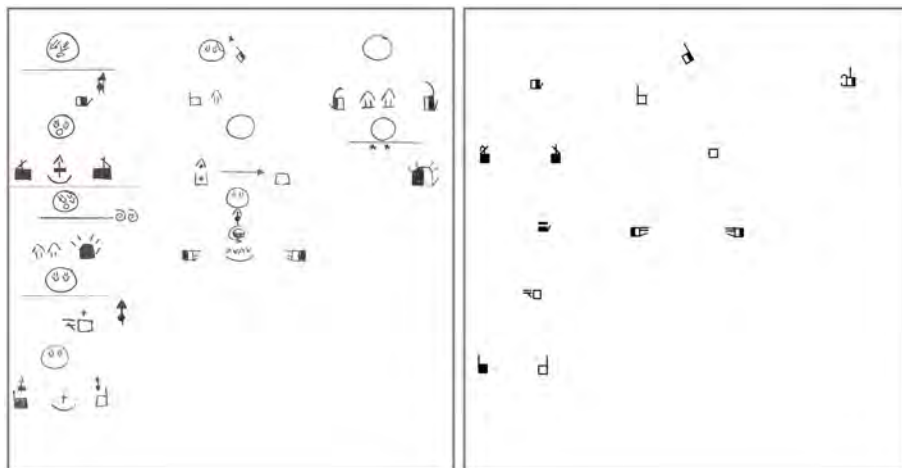


Figure 7.5: A handwritten page belonging to the DS-TEST-POITIERS dataset, as it appears before (left) and after (right) the recognition of the glyphs belonging to the Hand Configurations area.

It is also possible to observe a relevant percentage of failures in the recognition (15.3% for DS-TEST-PEAR and 13.2% for DS-TEST-POTIERS), this result was expected, since the recognition of the Hand Configurations is not completely over (See Tab. 6.6). In fact, the recognition of a number of shapes is yet to be developed, as a result, both the OGR area inference based on those shapes, and the consequent recognition, are not performed.

Finally, the strictness of the feature evaluations left no space for false positives.

Fig. 7.5 shows a handwritten page belonging to the DS-TEST-POITIERS dataset, as it appears before (left) and after (right) the recognition.

## 7.2 The SW-OGR Engine in action

As explained in Section 7.1, not every part of the SW-OGR is fully developed or tested. Some classes just perform the recognition of a restricted set of glyphs within their OGR areas, some classes need further debugging and testing.

Nonetheless, as the conclusion of the present chapter, it is worth showing that the SW-OGR Engine is already functional even if it is still under development. Please notice that the figures in the present section are not meant to provide an analysis of the recognition procedure from a quantitative point of view.

We picked a number of pages from both the DS-PEAR and the DS-POITIERS datasets, and we operated the recognition of any glyph supported by the SW-OGR Engine. The recognition was performed by allowing any class, tested and untested, to perform its evaluations for the recognition.

The results are shown in Fig 7.6 and Fig. 7.7. Recognition inaccuracies and failures can be spotted along successfully recognized glyphs within each figure.

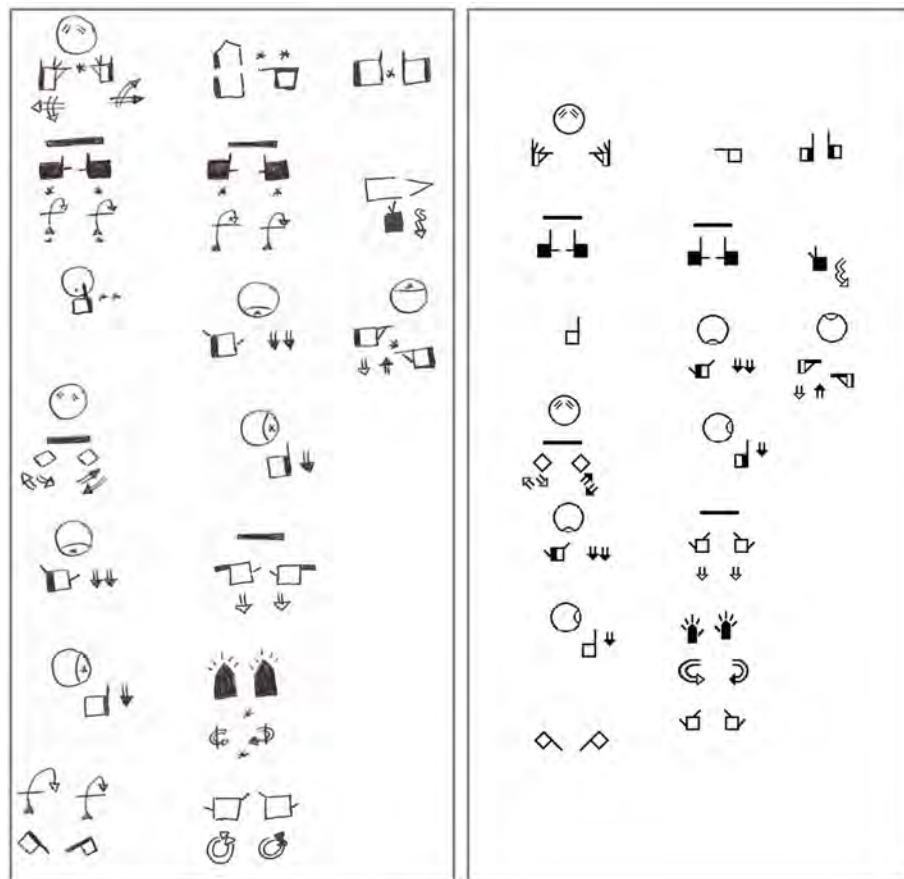


Figure 7.6: A handwritten page belonging to the DS-TEST-PEAR dataset, as it appears before (left) and after (right) the recognition.

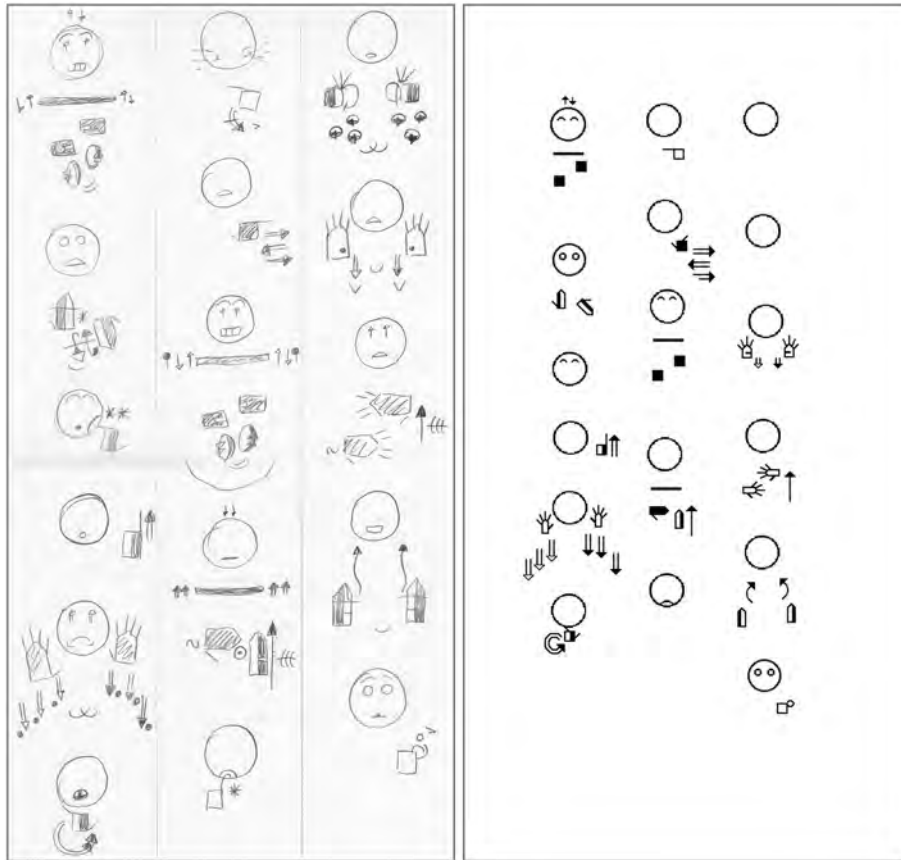


Figure 7.7: A handwritten page belonging to the DS-TEST-POITIERS dataset, as it appears before (left) and after (right) the recognition.



# Conclusions and future research

## Conclusions et recherche future en Français

Les défis que doivent relever au quotidien les sourds pour surmonter le fracture numérique qui les sépare du monde ont été la source d'inspiration de la présente étude. Vu et considéré que la plupart des problèmes dans ce domaine de la recherche ne sont - pour ainsi dire - jamais abordés, ces travaux proposent des études et des solutions au design d'accessibilité pour les sourds, fondé sur l'inclusion des LS. Bien que les LS soient une forme d'expression très répandue chez les sourds, leur importation dans les services et les contenus électroniques reste, la plupart du temps, discrétionnaire et limitée aux vidéos. Pourtant, l'interaction des sourds avec les artefacts numériques pourrait tirer de gros avantages d'un système d'écriture permettant de représenter des expressions et des contenus en LS de façon intuitive. Les présents travaux ne constituent qu'un élément d'un framework global visant à soutenir la communication des sourds à travers la plus naturelle de leur forme d'expression (la LS) et nos efforts se sont concentrés sur la gestion numérique de SignWriting qui est l'un des systèmes d'écriture des langues des signes les plus prometteurs. Notre but final est de projeter et mettre au point un système supportant la production et l'usage de ressources en SignWriting pour les sourds (SWord). L'ultime finalité de SWord est de rendre SignWriting efficacement exploitable en tant que moyen de communication et en tant que support d'apprentissage pour sourds, notamment dans le domaine du numérique. Etant donné que SWord est étroitement lié avec les LS, les études et les artefacts numériques illustrés dans ces travaux ont été conçus grâce à une synergie de compétences pluridisciplinaires. Ces compétences comprennent des linguistes, des interprètes, des experts en SignWriting et des psychologues.

Le premier logiciel incorporé dans le système SWord est un éditeur numérique pour SignWriting, à savoir un outil permettant de créer des ressources numériques en LS (signes ou histoires signées) écrites en SignWriting. Cet éditeur, baptisé SWift, est une véritable nouveauté par rapport à ses rivaux en raison de ses fonctionnalités de pointe permettant de seconder une composition naturelle et rapide de signes et d'histoires signées. Ces fonction-

nalités comprennent (la liste ne doit pas être considérée comme exhaustive): une composition guidée de signes, un support pour symboles ad hoc définis par les usagers et un système intuitif de recherche des symboles. SWift est conçu pour être incorporé dans n'importe quelle application/ressource web, afin d'apporter un support immédiat aux LS. En fait, SWift a déjà été intégré dans un environnement visuel de e-learning pour les sourds (DELE). En outre, SWift permet de mémoriser dans une banque de données tous les signes introduits par les usagers et tous les symboles les composant. Il est ainsi possible d'étudier des modèles de composition de signes et la co-occurrence des symboles dans des documents SW. C'est en travaillant, ces quatre dernières années, avec des linguistes que nous avons pris la mesure de l'importance des données statistiques issues de l'analyse des documents en SignWriting, vu qu'ils peuvent contribuer à comprendre les règles gouvernant les LS.

Les fonctionnalités de SWift et sa facilité d'utilisation ont été testées sur un échantillon de ses principaux usagers finals, à savoir des personnes connaissant et utilisant SignWriting. Etant donné que la majeure partie des participants aux tests étaient sourds, il a donc été nécessaire, pour obtenir des résultats fiables, d'utiliser une série d'outils et de méthodologies valables aussi bien pour les participants sourds que pour les entendants. Par conséquent, l'une des principales contributions qu'apportent les présents travaux à la Human-Computer Interaction est le protocole Think by Signs, à savoir une méthodologie de tests basée sur le protocole Think Aloud. Think by Signs, expressément conçu pour tester des applications sur des participants aussi bien sourds qu'entendants, est en mesure de fournir de précieuses informations aussi bien sur la facilité d'utilisation d'une application que sur la fiabilité de ses fonctionnalités. SWift a été testé avec succès à l'aide de différents outils, parmi lesquels figure le protocole Think by Signs. Les résultats du test se sont révélés particulièrement utiles pour résoudre une série de lacunes de l'application, que ce soit de simples défauts d'ergonomie que de véritables bugs.

Bien que de grandes avancées ont été accomplies dans le design, les éditeurs numériques pour SignWriting, tels que SWift, sont encore bien loin d'offrir à l'utilisateur une interface en mesure d'émuler la simplicité de l'écriture manuelle. C'est la raison pour laquelle nous avons conçu une nouvelle génération d'éditeurs numériques pour SignWriting afin d'épargner à l'utilisateur toutes les contraintes qu'impose une interface WIMP dans la composition d'un signe. Notre objectif était d'implémenter la qualité de l'interaction pour se rapprocher le plus possible de la méthode « papier-stylo » que tous les êtres humains utilisent habituellement pour écrire ou dessiner. Pour atteindre cet objectif, nous avons conçu une technique permettant de convertir électroniquement des images contenant des symboles SignWriting manuscrits (ou imprimés) en textes SignWriting numériques (codifiés dans le même format que celui de SWift).

Nous avons décidé de baptiser cette technique SignWriting Optical Glyph Recognition (SW-OGR). Le moteur de SW-OGR est en mesure de reconnaître

des textes SignWriting en se basant sur les caractéristiques structurelles (géométriques et topologiques) des symboles et sur leurs relations topologiques. Nous nous sommes également appuyés sur des informations intrinsèques au contexte, comme par exemple la connaissance de l'organisation de l'alphabet SignWriting. Dans ces travaux, nous avons exposé nos recherches sur l'alphabet SignWriting afin d'identifier les critères géométriques permettant de classer les symboles, d'élaborer une procédure de reconnaissance et de développer et tester le moteur SW-OGR. La codification de l'OGR est notamment l'une des majeures contributions de cette recherche puisqu'elle constitue (par rapport à l'ISWA) une codification nouvelle et différente se fondant exclusivement sur les caractéristiques structurelles des symboles. Par ailleurs, cette nouvelle codification n'est pas uniquement réservée à SW-OGR, elle peut être employée, en tant qu'organisation logique, à d'autres moteurs OGR et, plus en général, à n'importe quel système fondé sur une codification structurelle (plutôt que linguistique) des symboles de SignWriting. La phase du test d'application a prouvé que l'actuelle implémentation du moteur SW-OGR est valable, son opérationnalité a été mise à l'épreuve en utilisant une banque de données hétérogène contenant des textes rédigés par des personnes différentes dans des contextes différents.

Réduire la fracture numérique, qui marginalise les sourds, est un objectif ambitieux et il reste encore beaucoup à faire dans ce domaine, notamment dans le développement des fonctions du logiciel SWord. L'évolution naturelle de SWift impliquera très probablement la création d'une fonctionnalité d'écriture à main libre servant d'interface avec le moteur SW-OGR. Grâce à l'interaction des deux systèmes, les usagers de SignWriting bénéficieront d'une interface qu'ils connaissent déjà tout en utilisant une nouvelle fonctionnalité qui accélèrera considérablement le processus de composition des signes et des histoires signées. Par conséquent, le moteur SW-OGR doit être étendu pour contenir toutes les zones OGR (à savoir toutes les catégories de l'ISWA) dans le processus de reconnaissance. La conception de l'application a été portée à terme, des contrôles de routine structuraux ont été identifiés pour chaque catégorie de SignWriting, mais il faut encore s'efforcer de développer le concept pour arriver à un support complet à tous les symboles de SignWriting. Par ailleurs, comme il a déjà été proposé dans d'autres études sur l'Optical Music Recognition (OMR), SW-OGR pourrait être plus performant en intégrant (et en apprenant) des acteurs humains dans le processus de reconnaissance. Dans cette étude, l'acteur humain a été essentiellement présenté comme une figure ayant un rôle de réviseur dans le processus de reconnaissance, du fait qu'aucune reconnaissance ne peut être parfaitement exacte en toute circonstance. A l'avenir, le précieux feedback des révisions fournies par les acteurs humains pourrait être utilisé pour implémenter les fonctionnalités de l'apprentissage actif du système. SW-OGR pourrait conserver une trace des variations les plus fréquentes effectuées par les usagers et les appliquer automatiquement (ou à la demande) afin d'améliorer le résultat de la reconnaissance dans des délais graduellement décroissants pour l'acteur humain. Comme nous l'avons déjà évoqué

dans ces présents travaux, SW-OGR est un module d'application réutilisable qui peut être employé dans n'importe quel éditeur de SignWriting. Ces éditeurs peuvent varier du simple éditeur de base pour des applications mobiles ou pour le web à un éditeur professionnel avancé à usages divers, telle que la recherche ou l'authoring/publishing. Ces éditeurs, à leur tour, peuvent être incorporés dans une vaste gamme d'applications, notamment (mais pas exclusivement) la messagerie instantanée, les logiciels d'apprentissage, la productivité et le gaming. En général, un éditeur numérique pour SignWriting favorise la production et la diffusion de contenus accessibles aux sourds, ce qui est l'un de nos objectifs premiers. Le hardware plus approprié pour héberger une application basée sur SW-OGR devrait évidemment posséder des fonctionnalités tactiles, tels que les tablettes, les portables, les ordinateurs à écran tactile, etc. afin d'exploiter au mieux les avantages d'une interface naturelle pendant la composition d'un signe.

Un de nos objectifs majeurs est d'utiliser SignWriting pour transcrire les signes en prenant une vidéo signée comme flux d'entrée. Ceci implique l'usage de techniques de reconnaissance des formes pour transformer les gestes d'une personne produisant des signes en symboles SignWriting. Les positions et les mouvements capturés pourraient exploiter le système de codification OGR aussi bien pour la classification des caractéristiques que pour leur mémorisation. Les signes reconnus pourraient être facilement utilisés comme flux de données pour animer les avatars marquants grâce aux informations de production exhaustives que constitue chaque symbole SignWriting. Nous avons entamé notre recherche sur cette nouvelle méthode d'acquisition près le Département de Informatique de « Sapienza » Università di Roma.

Pour conclure, nous soulignerons que l'objectif final de nos travaux rejoint celui de SWord, à savoir promouvoir SignWriting en tant que moyen de communication performant pour les sourds dans l'univers numérique. Comme nous l'avons longuement exposé ci-dessus, grâce à nos recherches et au logiciel, que nous avons développé au cours de ces quatre dernières années, les résultats concrets de SWord sont désormais à portée de main. Ces résultats incluent la numérisation du corpus en SignWriting pour la communauté des sourds et pour celle des chercheurs spécialisés en linguistique des LS, des éditeurs numériques pour SignWriting (web et mobile) avec un support pour l'écriture manuscrite et des avatars en mesure de produire des signes à partir d'un texte SignWriting. Cette recherche veut être une réponse à la fracture numérique et notre contribution pour limiter son impact sur les sourds.

---

## Conclusions and future research in English

The present work has been significantly inspired by the challenges posed by the digital divide experienced by deaf people. Acknowledging that most issues in this research field remain largely unaddressed, the present work proposes studies and solutions for deaf-oriented accessibility design, based on SL inclusion. Even if SLs represent a widespread form of expression for deaf people, their inclusion in electronic services and contents is still limited, when available, to videos. However, the interaction of deaf people with digital artifacts may take great advantage from exploiting a written system to represent SL expressions and contents in an intuitive way.

The present work is a brick in the creation of an overall framework targeted at supporting deaf people through their most natural form of expression. In particular, we presented our efforts towards improving the computer handling of SignWriting, which is one of the most promising systems devised to write SLs. Our general goal is the design and the implementation of a framework to support the production and the use of SignWriting-oriented resources for the deaf (SWord). The ultimate purpose of SWord is to make SignWriting effectively exploitable as a communication mean and a learning support for deaf people, especially in the digital world. Since SWord deals directly with SLs, the studies and the digital artifacts presented in this work have been designed with the aid of a multi-disciplinary range of competences. Such competences include: linguists, interpreters, SignWriting experts and psychologists.

The first software to be included into the SWord framework was a SignWriting digital editor, i.e. a tool that enables the creation of digital SL resources (signs, or signed stories) written in SignWriting. Our editor is named SWift, and it represents an innovation with respect to its competitors, due its advanced features, aimed at supporting a natural and swift composition of signs and signed stories. Such features include (but they are not limited to) predictive text, support for user-defined *ad hoc* symbols, and an intuitive glyph search system. SWift is designed to be included into any web application/resource to provide a prompt SL support. Presently, SWift has already been integrated into a visual E-Learning environment for deaf people (DELE). Moreover, SWift allows to store composed signs in a database together with the component glyphs. This allows to study patterns and their co-occurrence in SW documents. Working with linguists in the past four years, we are well aware of the key importance of the derived statistical data, since they can help to gain understanding of the rules of SLs.

The features and the usability of SWift were tested with a sample of its main target users, i.e. people which are proficient with SignWriting. Most of the participants to the test were deaf, as a consequence, in order to gather reliable data, it was necessary to employ a set of tools and methodologies viable for both deaf and hearing participants. As consequence, one of the main contributions of the present work to the HCI field is represented by the *Think by Signs* protocol, i.e. a testing methodology based on the

Think-Aloud Protocol. Think by Signs is specifically designed to perform application testing with both deaf and hearing participants, and it is capable of providing valuable information about the level of usability and about the correct functioning of the features of any application. SWift has been successfully tested using, among other tools, the Think by Signs protocol; the results of the test were very useful to fix a number of issues with the applications, ranging from usability flaws to bugs.

Notwithstanding many design efforts, SignWriting digital editors, such as SWift, are still far from granting the user an interface which is able to emulate the simplicity of handwriting. As a consequence, we designed a new generation of SignWriting editors, able to lift the user of any burden related to clicking, dragging, searching, browsing on the UI during the composition process of a sign. Our goal was to implement an interaction style which is as similar as possible to the paper-pencil approach that humans normally use when writing or drawing. In order to achieve such goal, we designed a technique to operate the electronic conversion of images containing handwritten or printed SignWriting symbols into machine-encoded SignWriting texts (in the same format adopted by SWift).

Such technique is referred to as the SW-OGR. The SW-OGR engine is able to perform the recognition of SignWriting texts by only working with the structural (geometric and topological) features of the symbols, and with their topological relationships. We also relied on context-dependent information, such as the knowledge of the organization of the SignWriting alphabet. In this work we presented the studies performed on the SignWriting alphabet in order to identify structural criteria to classify its symbols, the design of the recognition procedure, and the development and testing of the SW-OGR engine. In particular, one of the main contributions of the present research is represented by the OGR coding, i.e. a new, different (with respect to ISWA) coding for SignWriting, based exclusively on the structural features of the symbols. Such coding is not SW-OGR-specific, it can be employed as underlying logical organization for other OGR engines, and, in general, within any system where a structural (rather than linguistic) coding for SignWriting symbols is required. The testing phase of the application demonstrated that the present implementation of the SW-OGR engine is currently working, and its validity has been proved with a heterogeneous group of datasets, containing texts written from different people, in different contexts.

Leveling the digital divide affecting deaf people is an ambitious goal, and much work is yet to be done in order to achieve it. In particular, regarding the SWord framework, some development work is still pending. The natural evolution of SWift shall most likely imply the development of a handwriting functionality intended to serve as an interface with the SW-OGR engine. Thanks to the integration of the two systems, SignWriting users will benefit from a known interface while exploiting a new functionality which can dramatically speed up the composition process of signs and signed stories. The SW-OGR needs to be expanded in order to include all OGR areas (in other words, all ISWA categories) into the recognition. The application is

---

fully designed, structural check routines have been identified for each OGR area, but a further development effort is needed, in order to achieve a full support for any glyph in SignWriting. Furthermore, as proposed in a number of other studies, mainly related to Optical Music Recognition (OMR), SW-OGR could benefit by including (and learning from) human actors in the recognition process. Throughout the present work, the role of the human actor has been presented mainly as one with recognition review responsibilities, due to the fact that no recognition can be perfectly accurate under any circumstances. In future, the valuable review feedback provided by the human actors could be employed to implement active learning capabilities for the system. SW-OGR shall keep track of the most frequent changes performed by the users and it shall apply them automatically (or on-demand) in order to improve the result of the recognition, with a gradually decreasing time cost for the human actor.

As already mentioned in the present work, SW-OGR is a reusable application module which could be employed within any SignWriting editor. Such editors may range from simple, minified editors targeted for mobile or web applications, to more complex, professional editors intended for different purposes, such as research or authoring/publishing. Such editors can be in turn embedded within a wide range of applications, including instant messaging, learning, productivity and gaming software. In general, a SignWriting editor fosters the production and the diffusion of deaf-accessible content, which is one of our goals. The most appropriate hardware to host SW-OGR-based application should provide touch capabilities, such as tablets, mobile phones, computers with touchscreens, etc. in order to fully exploit the advantages of a natural interface while composing a sign.

Another important goal we are pursuing is to use SignWriting to transcribe signs, taking a signed video as an input feed. This involves employing pattern-recognition techniques to map the gestures of a person producing signs, into SignWriting glyphs. The captured positions and movements could benefit of the features of the OGR coding system, both for feature classification and storage purposes (though the first purpose seems more productive). The recognized signs could easily be employed as data feed to animate avatars, due to the accurate production information carried by each SignWriting symbol. We already started our efforts to investigate and design this new acquisition method at “Sapienza” University of Rome.

In the end, it is our intention to remark that the ultimate goal of our work corresponds to the goal of the SWord framework, i.e. promoting SignWriting as an effective communication mean for deaf people in the digital world. As explained through the present work, thanks to the studies and to the software developed during the past four years, practical outcomes of the SWord framework are at hand. Such outcomes include digitized SignWriting corpora for the deaf and the linguistic research community, web and mobile SignWriting digital editors featuring handwriting support, and signing avatars able to produce signs taking a SignWriting text as input. This is our contribution in lessening the impact of the digital divide on deaf people.

## Conclusioni e ricerca futura in Italiano

Il presente lavoro è stato ispirato in modo molto significativo dalle sfide poste dal digital divide che le persone sorde sperimentano ogni giorno. Riconoscendo che la maggior parte dei problemi in questo campo di ricerca restano in gran parte non affrontati, il presente lavoro propone studi e soluzioni per il design di accessibilità per le persone sorde, basato sull'inclusione delle LS. Nonostante le SL rappresentino una forma di espressione molto diffusa tra i sordi, la loro inclusione in servizi e contenuti elettronici è ancora limitata, quando disponibile, all'utilizzo di video. Tuttavia, l'interazione delle persone sorde con gli artefatti digitali potrebbe trarre un grande vantaggio da un sistema di scrittura per rappresentare espressioni e contenuti in LS in modo intuitivo.

Il presente lavoro è solo un tassello nella creazione di un framework globale mirato a sostenere la comunicazione delle persone sorde attraverso la loro forma più naturale di espressione. In particolare, abbiamo presentato i nostri sforzi per migliorare la gestione digitale di SignWriting, che è uno dei sistemi di scrittura più promettenti per le LS. Il nostro obiettivo generale è la progettazione e la realizzazione di una sistema per supportare la produzione e l'uso di risorse in SignWriting per i sordi (SWord). Il fine ultimo di SWord è di rendere SignWriting efficacemente utilizzabile come mezzo di comunicazione e supporto di apprendimento per le persone sorde, in particolare nel mondo digitale. Del momento che SWord è strettamente legato con le LS, gli studi e gli artefatti digitali presentati in questo lavoro sono stati progettati grazie ad un range di competenze multidisciplinare. Tali competenze includono: linguisti, interpreti, esperti di SignWriting e psicologi.

Il primo software incluso nel sistema SWord è un editor digitale per SignWriting, vale a dire uno strumento che consente la creazione di risorse digitali in LS (segni, o storie segnate) scritte in SignWriting. Il nostro editor si chiama SWift, e rappresenta una novità rispetto ai suoi concorrenti grazie alle sue funzionalità avanzate, che hanno l'obiettivo di favorire una composizione naturale e rapida di segni e storie segnate. Tali funzionalità includono (ma non sono limitate a) composizione guidata di segni, supporto per simboli ad-hoc definiti dall'utente e un sistema intuitivo di ricerca dei simboli. SWift è progettato per essere incluso in qualsiasi applicazione/risorsa web, in modo da fornire un pronto supporto per le LS. In realtà, SWift è già stato integrato in un ambiente visivo di e-learning per le persone sorde (DELE). Inoltre, SWift permette di memorizzare in un database i segni inseriti dagli utenti, insieme ai singoli simboli che li compongono. Questo permette di studiare i pattern di composizione dei segni e la co-occorrenza dei simboli nei documenti SW. Lavorando con linguisti, negli ultimi quattro anni, abbiamo preso coscienza dell'importanza fondamentale dei dati statistici che possono derivare dall'analisi dei documenti in SignWriting, in quanto possono contribuire a comprendere le regole che governano le LS.

L'usabilità e le funzionalità di SWift sono state testate con un campione dei suoi principali utenti finali, cioè con persone che conoscono SignWriting. La



maggior parte dei partecipanti al test erano sordi, di conseguenza, al fine di raccogliere risultati affidabili, è stato necessario adottare una serie di strumenti e di metodologie valide sia per partecipanti sordi, sia per partecipanti udenti. Di conseguenza, uno dei principali contributi del presente lavoro al campo della HCI è rappresentato dal protocollo Think by Signs, ossia una metodologia di test basata sul protocollo Think-Aloud. Think by Signs è specificamente progettato per testare applicazioni con partecipanti sia sordi che udenti, ed è in grado di fornire preziose informazioni circa il livello di usabilità e sul corretto funzionamento delle funzionalità di un'applicazione. SWift è stato testato con successo con diversi strumenti, tra i quali il protocollo Think by Signs; i risultati del test sono stati molto utili per risolvere una serie di problemi dell'applicazione, a partire da difetti di usabilità fino a veri e propri bug.

Nonostante molti avanzamenti nel design, gli editor digitali per SignWriting, come SWift, sono ancora lontani dal fornire all'utente un'interfaccia che sia in grado di emulare la semplicità della scrittura manuale. Di conseguenza, abbiamo progettato una nuova generazione di editor digitali per SignWriting, in grado di sollevare l'utente da tutti gli oneri relativi all'utilizzo di un'interfaccia WIMP durante la composizione di un segno. Il nostro obiettivo era quello di implementare uno stile di interazione che fosse il più simile possibile al metodo "carta e penna" che gli esseri umani normalmente utilizzano durante la scrittura o il disegno. Al fine di raggiungere tale obiettivo, abbiamo progettato una tecnica il cui scopo è quello di operare la conversione elettronica di immagini contenenti simboli SignWriting scritti a mano (o stampati) in testi SignWriting digitali (codificati nello stesso formato adottato da SWift).

Abbiamo deciso di chiamare tale tecnica SignWriting Optical Glyph Recognition (SW-OGR). Il motore SW-OGR è in grado di effettuare il riconoscimento di testi SignWriting basandosi sulle caratteristiche strutturali (geometriche e topologiche) dei simboli, e sulle relazioni topologiche tra di essi. Abbiamo anche potuto fare affidamento su informazioni dipendenti dal contesto, come ad esempio la conoscenza dell'organizzazione dell'alfabeto SignWriting. In questo lavoro abbiamo presentato gli studi effettuati sull'alfabeto SignWriting al fine di identificare i criteri geometrici per classificare i suoi simboli, la progettazione della procedura di riconoscimento, e lo sviluppo e di test del motore SW-OGR. In particolare, uno dei principali contributi della presente ricerca è rappresentato dalla codifica OGR, cioè una nuova, differente (rispetto a ISWA) codifica per SignWriting, basata esclusivamente sulle caratteristiche strutturali dei simboli. Tale codifica non è specifica per SW-OGR, può essere impiegata come organizzazione logica per altri motori OGR, e, in generale, in qualsiasi sistema in cui sia richiesta una codifica strutturale (piuttosto che linguistica) per i simboli di SignWriting. La fase di test dell'applicazione ha dimostrato che l'attuale implementazione del motore SW-OGR è valida, la sua operatività è stata messa alla prova usando un dataset eterogeneo, contenente testi scritti da persone diverse, in contesti diversi.

Ridurre il digital divide che colpisce le persone sorde è un obiettivo ambizioso, e molto lavoro è ancora da fare per raggiungerlo. In particolare, per quanto riguarda SWord, è ancora necessario uno sforzo di sviluppo software. L'evoluzione naturale di SWift comporterà molto probabilmente lo sviluppo di una funzionalità di scrittura a mano libera destinata a fungere da interfaccia con il motore SW-OGR. Grazie all'integrazione dei due sistemi, gli utenti di SignWriting beneficeranno di un'interfaccia già nota, sfruttando una nuova funzionalità in grado di accelerare notevolmente il processo di composizione di segni e storie segnate. Il motore SW-OGR deve essere ampliato per includere tutte le aree OGR (in altre parole, tutte le categorie ISWA) nel riconoscimento. L'applicazione è completamente progettata, routine di controllo strutturale sono state identificate per ciascuna categoria SignWriting, ma è necessario un ulteriore sforzo di sviluppo, al fine di ottenere un supporto completo per qualsiasi simbolo SignWriting. Inoltre, come proposto in una serie di altri studi, principalmente riguardanti la Optical Music Recognition (OMR), SW-OGR potrebbe ricavare molti vantaggi includendo (e imparando da) attori umani nel processo di riconoscimento. In tutto il presente lavoro, il ruolo dell'attore umano è stata presentato principalmente come un ruolo con responsabilità di revisione del riconoscimento, a causa del fatto che nessun riconoscimento può essere perfettamente accurato in ogni circostanza. In futuro, il prezioso feedback delle revisioni fornite dagli attori umani potrebbe essere impiegate per implementare funzionalità di apprendimento attivo per il sistema. SW-OGR potrebbe tener traccia delle più frequenti variazioni effettuate dagli utenti e potrebbe applicarli automaticamente (o su richiesta), al fine di migliorare il risultato del riconoscimento, con un costo di tempo gradualmente decrescente per l'attore umano. Come già accennato nel presente lavoro, SW-OGR è un modulo applicativo riutilizzabile che può essere impiegato in qualsiasi editor di SignWriting. Tali editor possono variare da semplici, essenziali editor per applicazioni mobile o web, a più complessi editor professionali destinati ad usi diversi, come ricerca o authoring/publishing. Tali editor possono essere a loro volta incorporati all'interno di una vasta gamma di applicazioni, tra le quali (ma non solo) l'instant messaging, software per l'apprendimento, la produttività e il gaming. In generale, un editor digitale per SignWriting favorisce la produzione e la diffusione di contenuti accessibili ai sordi, che è uno dei nostri obiettivi. L'hardware più appropriato per ospitare un'applicazione basata su SW-OGR dovrebbe naturalmente fornire funzionalità touch, come tablet, telefoni cellulari, computer con touchscreen, ecc, al fine di sfruttare appieno i vantaggi di un'interfaccia naturale durante la composizione di un segno.

Un altro obiettivo importante che stiamo perseguendo è quello di utilizzare SignWriting per trascrivere i segni prendendo un video segnato come feed di ingresso. Ciò comporta l'impiego di tecniche di pattern recognition per trasformare i gesti di una persona che produce segni in simboli SignWriting. Le posizioni e movimenti catturati potrebbero beneficiare del sistema di codifica OGR, sia per la classificazione delle caratteristiche, sia per la loro mem-

orizzazione. I segni riconosciuti potrebbero essere facilmente utilizzati come feed di dati per animare gli avatar segnanti, a causa delle accurate informazioni di produzione che ogni simbolo SignWriting rappresenta. Abbiamo già dato inizio ai nostri sforzi per progettare questo nuovo metodo di acquisizione presso il Dipartimento di Informatica dell'Università "Sapienza" di Roma.

Per concludere, è nostra intenzione sottolineare che il fine ultimo del nostro lavoro corrisponde all'obiettivo di SWord, cioè promuovere SignWriting come mezzo di comunicazione efficiente per le persone sorde nel mondo digitale. Come spiegato nel presente lavoro, grazie agli studi e al software sviluppato nel corso degli ultimi quattro anni, i risultati pratici di SWord sono a portata di mano. Tali risultati comprendono digitalizzare corpora SignWriting per la comunità sorda e per quella dei ricercatori in linguistica delle LS, editor digitali per SignWriting (web e mobile) con supporto per la scrittura a mano, e avatar in grado di produrre i segni a partire da un testo SignWriting. Questo, in ultima analisi, è il nostro contributo nel ridurre l'impatto del digital divide sulle persone sorde.



# Glossary

**frag** A 8-connected foreground component within a slice..

**slice** A section of a text. If the text follows a horizontal visual organization, a slice is a row within the text. Otherwise, if the text follows a vertical visual organization, a slice is a column within the text..

**text** An image containing handwritten or printed SignWriting glyphs.



# Acronyms

**ASL** American Sign Language.

**CNR** National Research Council.

**DELE** Deaf-oriented E-Learning Environment.

**DOM** Document Object Model.

**FSL** Flemish sign Language.

**GFTT** Good Features To Track.

**HamNoSys** Hamburg Notation System.

**HCI** Human-Computer Interaction.

**IPA** International Phonetic Alphabet.

**ISTC** Institute of Cognitive Sciences and Technologies.

**ISWA** International SignWriting Alphabet.

**ISWA-2008** International SignWriting Alphabet 2008.

**ISWA-2010** International SignWriting Alphabet 2010.

**ISWA-BIANCHINI** International SignWriting Alphabet Bianchini.

**LIS** Italian Sign Language.

**LSF** French Sign Language.

**MIT** Massachusetts Institute of Technology.

**NAD** National Association of the Deaf.

**NN** Neural Network.

**OCR** Optical Character Recognition.

**OMR** Optical Music Recognition.

**POI** Point of Interest.

**QUIS** Questionnaire for User Interaction Satisfaction.

**SCSI** Small Computer System Interface.

**SL** Sign Language.

**SLDS** Sign Language and Deaf Studies.

**SM** SignMaker.

**SW-OGR** SignWriting Optical Glyph Recognition.

**SWift** SignWriting improved fast transcriber.

**SWML** SignWriting Markup Language.

**SWord** SignWriting-oriented resources for the deaf.

**TAP** Think-Aloud Protocol.

**UI** User Interface.

**VL** Vocal Language.

**W3C** World Wide Web Consortium.

**WCAG** Web Content Accessibility Guidelines.

**WIMP** Windows Icons Menu Pointers.



# Bibliography

- Abdelazim, H., Mousa, A., Saleh, Y., & Hashish, M. (1990). Arabic text recognition using partial observation approach. In *Proceedings of the 12th National Computer Conference* (p. 427—437). Riyadh, Saudi Arabia.
- Abuhaiba, I., Holt, M., & Datta, S. (1998). Recognition of off-line cursive handwriting. *Computer Vision and Image Understanding*, *64*(1), 19–138.
- Ahmed, P., & Al-Ohali, Y. (2000). Arabic Character Recognition: Progress and Challenges. *Journal of King Saud University - Computer and Information Sciences*, *12*(0), 85–116. doi: [http://dx.doi.org/10.1016/S1319-1578\(00\)80004-X](http://dx.doi.org/10.1016/S1319-1578(00)80004-X)
- Al-Badr, B., & Haralick, B. (1996). Segmentation-free recognition of Arabic text. In *Proceedings of the The 5th International Conference and Exhibition on Multi-Lingual Computing*. Cambridge, United Kingdom: Cambridge University Press.
- Al-Badr, B., & Mahmoud, S. (1995). Survey and bibliography of Arabic optical text recognition. *Signal Processing*, *41*(1), 49–77. doi: [http://dx.doi.org/10.1016/0165-1684\(94\)00090-M](http://dx.doi.org/10.1016/0165-1684(94)00090-M)
- Altuwaijri, M., & Bayoumi, M. (1994). Arabic text recognition using neural networks. In *Proceedings of IEEE International Symposium on Circuits and Systems* (pp. 415–418). London, United Kingdom.
- Amin, A. (1998). Off-line Arabic character recognition: the state of the art. *Pattern Recognition*, *31*(5), 517–530. doi: [http://dx.doi.org/10.1016/S0031-3203\(97\)00084-8](http://dx.doi.org/10.1016/S0031-3203(97)00084-8)
- Amin, A., & Masini, G. (1986). Machine recognition of multi font printed arabic texts. In *Proceedings of the 8th International Joint Conference on Pattern Recognition* (p. 392—395). Paris, France.
- Antinoro Pizzuto, E. (2009). Meccanismi di coesione testuale e Strutture di Grande Iconicità nella Lingua dei segni Italiana (LIS). In B. C. & C. A. (Eds.), *Alcuni capitoli della grammatica della LIS* (pp. 130–151). Venice, Italy: Libreria Editrice Cafoscarina.
- Antinoro Pizzuto, E., Bianchini, C., Capuano, D., Gianfreda, G., & Rossini, P. (2010). Language resources and visual communication in a deaf centered multimodal e-learning environment: issues to be addressed. In *Proceedings of the first Workshop on Supporting eLearning with Language Resources and Semantic Data, Seventh Interna-*

- tional Conference on Language Resources and Evaluation* (pp. 18–23). Retrieved on Retrieved November 5, 2013 from <http://www.lrec-conf.org/proceedings/lrec2010/workshops/W15.pdf>.
- Antinoro Pizzuto, E., & Pietrandrea, P. (2001). The notation of signed texts: Open questions and indications for further research. In B. Bergman, P. Boyes-Braem, T. Hanke, & E. Antinoro Pizzuto (Eds.), *Sign Transcription and Database Storage of Sign Information, special issue of Sign Language and Linguistics* (Vol. 4, pp. 29–45). Amsterdam, Netherlands: John Benjamins.
- Antinoro Pizzuto, E., Rossini, P., & Russo, T. (2006). Representing signed languages in written form: questions that need to be posed. In *Proceedings of the second "Workshop on the Representation and Processing of Sign Languages: lexicographic matters and didactic scenario"*, *International Conference on Language Resources and Evaluation - LREC 2006* (pp. 1–6). Paris, France: ELRA.
- ASL Font: Ways to write ASL*. (2013). Retrieved August 21, 2014, from <http://aslfont.github.io/Symbol-Font-For-ASL/ways-to-write.html>.
- Bazzi, I., Schwartz, R., & Makhoul, J. (1999). An omnifont open-vocabulary OCR system for English and Arabic. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(6), 495–504.
- Bellini, P., Bruno, I., & Nesi, P. (2001). Optical music sheet segmentation. In *Proceedings of the First International Conference on Web Delivering of Music* (pp. 183–190). doi: 10.1109/WDM.2001.990175
- Bianchini, C. (2012). *Analyse métalinguistique de l'émergence d'un système d'écriture des Langues des Signes: SignWriting et son application à la Langue des Signes Italienne (LIS)* (Unpublished doctoral dissertation). Université de Paris 8, Paris, France.
- Bianchini, C., & Borgia, F. (2012). Writing Sign languages: analysis of the evolution of the SignWriting system from 1995 to 2010, and proposals for future developments. In *Proceedings of the International Jubilee Congress, Technical University of Varna* (pp. 118–123).
- Bianchini, C., Borgia, F., Bottoni, P., & De Marsico, M. (2012). SWift: a SignWriting improved fast transcriber. In G. Tortora, S. Levialdi, & M. Tucci (Eds.), *Proceedings of the International Working Conference on Advanced Visual Interfaces* (p. 390-393). New York, NY: ACM. doi: <http://doi.acm.org/10.1145/2254556.2254631>
- Bianchini, C., Borgia, F., & Castelli, M. (2011, May). *Le "bidouillage": comment les locuteurs de Langue des Signes Italienne (LIS) s'approprient et modifient le SignWriting pour le rendre plus adapté à leurs exigences*. Paper presented at 25ème Colloque International du CerLiCO: Transcrire, écrire, formaliser - 2, Orleans, France.
- Bianchini, C., Borgia, F., & De Marsico, M. (2012). SWift - A SignWriting Editor to Bridge between Deaf World and E-learning. In *2013 IEEE 13th International Conference on Advanced Learning Technologies* (Vol. 0, p. 526-530). Los Alamitos, CA: IEEE Computer Society. doi: <http://doi.ieeecomputersociety.org/10.1109/ICALT.2012.235>

- Biblica. (2011). *Holy Bible: New International Version*. Elizabethton, TN: STL Distribution.
- Boost c++ libraries* (Tech. Rep.). (2004). Retrieved March 23, 2014, from <http://www.boost.org/>.
- Borgia, F. (2010). *SWift: SignWriting improved fast transcriber* (Unpublished master's thesis). Sapienza Università di Roma. Rome, Italy.
- Borgia, F. (2013). *Circle Detection - Reliability Assessment*. Retrieved on June 25, 2014, from <http://stackoverflow.com/questions/20165729/circle-detection-reliability-assessment>.
- Borgia, F., Bianchini, C., & De Marsico, M. (2014). Towards Improving the e-learning Experience for Deaf Students: e-LUX. In C. Stephanidis & M. Antona (Eds.), *Lecture Notes in Computer Science: Universal Access in Human-Computer Interaction* (Vol. 8514, p. 221-232). Berlin, Germany: Springer. doi: 10.1007/978-3-319-07440-5\_21
- Bottoni, P., Borgia, F., Buccarella, D., Capuano, D., De Marsico, M., & Labella, A. (2013). Stories and signs in an e-learning environment for deaf people. *Universal Access in the Information Society*, 12(4), 369–386. doi: 10.1007/s10209-012-0283-y
- Bottoni, P., Capuano, D., De Marsico, M., & Labella, A. (2012). DELE framework: an innovative sight on didactics for deaf people. *Journal of e-learning and Knowledge Society*, 8(3), 165–174.
- Bousslama, F. (1996). Neuro-fuzzy techniques in the recognition of written arabic characters. In *Proceedings of North American Fuzzy Information Process* (p. 142–146). Berkeley, CA.
- Bulwer, J. (1644). *Chirologia: or the naturall language of the hand. Composed of the speaking motions, and discoursing gestures thereof. Whereunto is added Chironomia: or, the art of manuell rhetoricke. Consisting of the naturall expressions, digested by art in the hand, as the chiefest instrument of eloquence*. London, United Kingdom: Thomas Harper.
- Caldwell, B. and Cooper, M. and Guarino Reid, L. and Vanderheiden, G. (Ed.). (2008). *Web Content Accessibility Guidelines 2.0 (WCAG 2.0)*. Retrieved November 11, 2013, from <http://www.w3.org/TR/WCAG>.
- Cavender, A., Trewin, A., & Hanson, V. (2009). *General Writing Guidelines for Technology and People with Disabilities*. Retrieved on November 12, 2013, from <http://www.sigaccess.org/welcome-to-sigaccess/resources/accessible-writing-guide>.
- Chafe, W. (1980). *The Pear Stories: Cognitive, Cultural, and Linguistic Aspects of Narrative Production*. New York, NY: Ablex.
- Channon, R., & van der Hulst, H. (2010). Notation Systems. In D. Brentari (Ed.), *Sign Languages*. Cambridge, United Kingdom: Cambridge University Press.
- Chaudhuri, B., & Pal, U. (1998). A complete printed Bangla OCR system. *Pattern Recognition*, 31(1), 531–549.
- Chin, J., Diehl, V., & Norman, K. (1998). Development of an Instrument

- Measuring User Satisfaction of the Human-Computer Interface. In E. Soloway, D. Frye, & S. B. Sheppard (Eds.), *Proceedings of ACM CHI'88 Conference on Human Factors in Computing Systems* (pp. 213–218). New York, NY: ACM.
- Choudhury, G. S., Droetboom, M., Dilauro, T., Fujinaga, I., & Harrington, B. (2000). Optical Music Recognition System within a Large-Scale Digitization Project. In *Proceedings of the International Society for Music Information Retrieval*.
- Curran, K., Woods, D., & O'Riordan, B. (2006). Investigating text input methods for mobile phones. *Telematics and Informatics*, 23(1), 1–21. doi: <http://dx.doi.org/10.1016/j.tele.2004.12.001>
- Cuxac, C. (1996). *Fonctions et structures de l'iconicité des Langues des Signes*. Thèse d'État. Université Paris V, Paris.
- Cuxac, C. (2000). La Langue des Signes Française (LSF): les voies de l'iconicité. In C. Cuxac (Ed.), *Faits de langues*. Paris, France: Ophrys.
- Danby, H. (Ed.). (1933). *The Mishnah*. Wotton-under-Edge, United Kingdom: Clarendon Press.
- Devroye, L., Györfi, L., & Lugosi, G. (1996). *A Probabilistic Theory of Pattern Recognition*. U.S. Government Printing Office.
- Di Renzo, A., Gianfreda, G., Lamano, L., Lucioli, T., Pennacchi, B., Rossini, P., ... Antinoro Pizzuto, E. (2012). *Scrivere la LIS con il SignWriting: manuale introduttivo*. Rome, Italy: ISTC-CNR.
- Di Renzo, A., Lamano, L., Lucioli, T., Pennacchi, B., & Ponzo, L. (2006). Italian Sign Language (LIS): Can We Write it and Transcribe it with SignWriting? In *Proceedings of the second "Workshop on the Representation and Processing of Sign Languages: lexicographic matters and didactic scenario"*, *International Conference on Language Resources and Evaluation - LREC 2006* (pp. 11–16). Paris, France: ELRA.
- Duda, R., & Hart, P. (1972). Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Communications of the ACM*, 15(1), 11–15. doi: 10.1145/361237.361242
- Encrevé, F. (2013). *Les sourds dans la société française au XIXe siècle: Idée de progrès et langue des signes*. Paris, France: Créaphis éditions.
- Eriksson, P., & Schmale, J. (1998). *The history of deaf people: A source book*. Örebro, Sweden: SIH Läromedel.
- Fajardo, I., Parra, E., Cañas, J., Abascal, J., & Lopez, J. (2008, July). *Web information search in Sign Language*. Paper presented at Informal Learning on the Web: Individual Differences and Evaluation Processes (Symposium), XXIX International Congress of Psychology, Berlin, Germany.
- Fajardo, I., Vigo, M., & Salmerón, L. (2009). Technology for supporting web information search and learning in Sign Language. *Interacting with Computers*, 21(4), 243–256. doi: 10.1016/j.intcom.2009.05.005
- Fehri, M., & Ahmed, M. (1994). *A new approach to arabic character recognition in multi-font document*. Paper presented at the 4th International Conference and Exhibition on Multi-Lingual Computing, Cambridge,

- United Kingdom.
- Fels, D., Richards, J., Hardman, J., & Lee, D. (2006). Sign language Web pages. *American Annals of the Deaf*, 151(4), 423–433.
- Franciosi, L. (2014). *Riconoscimento di glifi di movimento della mano nel SignWriting scritto a mano* (Unpublished bachelor's thesis). Sapienza Università di Roma. Rome, Italy.
- Fujinaga, I. (2005). Staff detection and removal. In S. E. George (Ed.), (pp. 1–39). IRM Press.
- Gales, M., & Young, S. (2007). The Application of Hidden Markov Models in Speech Recognition. *Foundations and Trends in Signal Processing*, 1(3), 195–304. doi: 10.1561/20000000004
- Garcia, B. (2006). The Methodological, Linguistic and Semiological Bases for the Elaboration of a Written Form of French Sign Language (LSF). In *Proceedings of the second "Workshop on the Representation and Processing of Sign Languages: lexicographic matters and didactic scenario"*, International Conference on Language Resources and Evaluation - LREC 2006 (pp. 31–36). Paris, France: ELRA.
- Garcia, B., & Sallandre, M. (2013). Transcription systems for sign languages: a sketch of the different graphical representations of sign language and their characteristics. In C. Müller, A. Cienki, E. Fricke, S. Ladewig, D. McNeill, & T. S. (Eds.), *Handbook 'Body-Language-Communication'* (pp. 1125–1338). Mouton De Gruyter.
- Göcke, R. (2003). Building a system for writer identification on handwritten music scores. In *Proceedings of the IASTED International Conference on Signal Processing, Pattern Recognition, and Applications* (pp. 250–255).
- Gonzalez, R., & Woods, R. (2011). *Digital Image Processing*. Cambridge University Press.
- Goraine, H., & Usher, M. (1994). Printed arabic text recognition. In *Proceedings of the 4th International Conference and Exhibition on Multi-Lingual Computing*. Cambridge, United Kingdom: Cambridge University Press.
- Hanke, T., & Storz, J. (2006). iLex - a database tool integrating sign language corpus linguistics and sign language lexicography. In *Proceedings of the second "Workshop on the Representation and Processing of Sign Languages: construction and exploitation of Sign Language corpora"*, International Conference on Language Resources and Evaluation - LREC 2008 (pp. 64–67). Paris, France: ELRA.
- Harris, C., & Stephens, M. (1988). A combined corner and edge detector. In *Proceedings of the Fourth Alvey Vision Conference* (pp. 147–151).
- Hassibi, K. (1994). Machine-printed Arabic OCR using neural networks. In *Proceedings of the 4th International Conference and Exhibition on Multi-lingual Computing (Arabic and Roman Script)* (pp. 2.3.1–2.3.12). London, UK: Cambridge.
- Holt, C., Stewart, A., Clint, M., & Perrott, R. (1987). An improved parallel thinning algorithm. *Communications of the ACM*, 30(2), 156–160.

- Hough, P. (1959). Machine Analysis Of Bubble Chamber Pictures. *Proceedings of the 2nd International Conference On High-Energy Accelerators, C590914*, 554–558.
- Itseez (Tech. Rep.). (2005). Retrieved March 23, 2014, from <http://itseez.com>.
- Itseez. (2014). *Opencv* (Tech. Rep.). Retrieved November 15, 2013, from <http://opencv.org/>.
- Jain, A. (1989). *Fundamentals of Digital Image Processing*. Prentice Hall.
- Karpov, A., & Ronzhin, A. (2014). A Universal Assistive Technology with Multimodal Input and Multimedia Output Interfaces. In C. Stephanidis & M. Antona (Eds.), *Universal Access in Human-Computer Interaction. Design and Development Methods for Universal Access* (Vol. 8513, p. 369-378). Springer International Publishing. doi: 10.1007/978-3-319-07437-5\_35
- Khorsheed, M. S. (2002). Off-Line Arabic Character Recognition – A Review. *Pattern Analysis & Applications*, 5(1), 31-45. doi: 10.1007/s100440200004
- Kipp, M., Nguyen, Q., Heloir, A., & Matthes, S. (2011). Assessing the Deaf User Perspective on Sign Language Avatars. In *Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility* (pp. 107–114). New York, NY: ACM. doi: 10.1145/2049536.2049557
- Kyle, J., Woll, B., & Pullen, G. (1988). *Sign Language: The Study of Deaf People and Their Language*. Cambridge, United Kingdom: Cambridge University Press.
- Lam, L., Lee, S., & Suen, C. (1992). Thinning Methodologies-A Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9), 869–885. doi: 10.1109/34.161346
- Lane, H. (1984). *When the Mind Hears: A History of the Deaf*. New York, NY: Random House.
- Lang, H., & Meath-Lang, B. (1995). *Deaf persons in the arts and sciences: A biographical dictionary*. Westport, CT: Greenwood Press.
- Lewis, C. (1982). *Using the "thinking aloud" method in cognitive interface design*. Yorktown Heights, NY: IBM T.J. Watson Research Center.
- Lewis, C., & Rieman, J. (1993). *Task-centered user interface design: A practical introduction*. University of Colorado, USA.
- Lynch, T. (1985). *Data Compression: Techniques and Applications*. Charlottesville, VA: Lifetime Learning Publications.
- MacMillan, K., Droettboom, M., & Fujinaga, I. (2002). Gamera: Optical music recognition in a new shell. In *Proceedings of the international computer music conference* (p. 482–485).
- Makhoul, J., Schwartz, R., Lapre, C., & Bazzi, I. (1998). A script-independent methodology for optical character recognition. *Pattern Recognition*, 31(9), 1285–1294.
- Margner, V., & El Abed, H. (2007). Arabic Handwriting Recognition Competition. In *Ninth international conference on document anal-*

- ysis and recognition* (Vol. 2, pp. 1274–1278). doi: 10.1109/ICDAR.2007.4377120
- Marschark, M., & Spencer, P. (2011). *The Oxford Handbook of Deaf Studies, Language, and Education, Second Edition* (Vol. 1). New York, NY: Oxford University Press.
- Miyao, H., & Nakano, Y. (1995). Head and stem extraction from printed music scores using a neural network approach. In *Proceedings of the Third International Conference on Document Analysis and Recognition* (Vol. 2, pp. 1074–1079). doi: 10.1109/ICDAR.1995.602095
- Mohanti, S. (1998). Pattern recognition in alphabets of Oriya language using Kohonen neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 12, 1007–1015.
- Mommsen, T., Krueger, P., & Kroll, W. (Eds.). (2010). *Corpus Iuris Civilis*. Swindon, United Kingdom: Bertram.
- Nakariakov, S. (2013). *The Boost C++ Libraries: Generic Programming*. CreateSpace Independent Publishing Platform.
- Niblack, W. (1985). *An Introduction to Digital Image Processing*. Birkerød, Denmark: Strandberg Publishing Company.
- Norman, D., & Draper, S. (1986). *User Centered System Design*. Mahwah, NJ: Lawrence Erlbaum Associates.
- Ong, W. (2002). *Orality and Literacy: The Technologizing of the Word*. London, United Kingdom: Routledge.
- Otsu, N. (1979). A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1), 62–66. doi: 10.1109/TSMC.1979.4310076
- Pal, U., & Chaudhuri, B. (2004). Indian script character recognition: a survey. *Pattern Recognition*, 37(9), 1887–1899. doi: <http://dx.doi.org/10.1016/j.patcog.2004.02.003>
- Papadias, D., Sellis, T., Theodoridis, Y., & Egenhofer, M. (1995). Topological Relations in the World of Minimum Bounding Rectangles: A Study with R-trees. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data* (pp. 92–103). New York, NY: ACM. doi: 10.1145/223784.223798
- Parker, J. (2010). *Algorithms for image processing and computer vision*. John Wiley & Sons.
- Parvez, M., & Mahmoud, S. (2013). Arabic handwriting recognition using structural and syntactic pattern attributes. *Pattern Recognition*, 46(1), 141–154. doi: <http://dx.doi.org/10.1016/j.patcog.2012.07.012>
- Perfetti, C., & Sandak, R. (2000). Reading optimally builds on spoken language. *Journal of Deaf Studies and Deaf Education*, 5, 32–50.
- Pirolli, P., & Card, S. (1999). Information foraging. *Psychological Review*, 106(4), 643–675.
- Presneau, J. (1998). *Signes et institution des sourds: XVIIIe-XIXe siècle*. Seyssel, France: Champ Vallon.
- Prillwitz, S., Leven, R., Zienert, H., Hanke, H., & Henning, J. (1989). *Hamburg Notation System for Sign Languages: an introductory guide, Ham-*

- NoSys version 2.0*. Seedorf, Germany: Signum.
- Pugin, L., Ashley, J., & Fujinaga, I. (2007.). MAP adaptation to improve optical music recognition of early music documents using Hidden Markov Models. In *Proceedings of the 8th International Society for Music Information Retrieval* (p. 513—516).
- Radutzky, E. (1993). The education of deaf people in Italy and the use of Italian Sign Language. In J. Van Cleve (Ed.), *Deaf history unveiled: Interpretations from the new scholarship* (p. 237—251). Washington, DC: Gallaudet University Press.
- Rebelo, A., Fujinaga, I., Paszkiewicz, F., Marcal, A., Guedes, C., & Cardoso, J. (2012). Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval*, 1(3), 173–190. doi: 10.1007/s13735-012-0004-6
- Ree, J. (1999). *I see a voice: Deafness, language and the senses — A philosophical history*. Metropolitan Books.
- Roberts, V., & Fels, D. (2006). Methods for inclusion: Employing think aloud protocols in software usability studies with individuals who are deaf. *International Journal of Human-Computer Studies*, 64(6), 489–501. doi: <http://dx.doi.org/10.1016/j.ijhcs.2005.11.001>
- Rotiroti, M. (2014). *Binarizzazione di documenti prodotti in SignWriting scritti a mano* (Unpublished bachelor's thesis). Sapienza Università di Roma. Rome, Italy.
- Sampson, G. (1990). *Writing Systems: A Linguistic Introduction*. Stanford, CA: Stanford University Press.
- Sanossian, H. (1996). An Arabic character recognition system using neural network. In *Proceedings of 1996 IEEE Signal Processing Society Workshop* (p. 340—348). IEEE.
- Sarfraz, M. (2005). *Computer-Aided Intelligent Recognition Techniques and Applications*. New York, NY: Wiley.
- Sauvola, J. and Pietikäinen, M. (2000). Adaptive document image binarization. *Pattern Recognition*, 33, 225–236.
- Sedley, D. (Ed.). (2003). *Plato's Cratylus*. Cambridge, United Kingdom: Cambridge University Press.
- Sezgin, M., & Sankur, B. (2004). Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1), 146–168. doi: 10.1117/1.1631315
- Shi, J., & Tomasi, C. (1994). Good features to track. In *Proceedings of the conference on computer vision and pattern recognition, 1994*. doi: 10.1109/CVPR.1994.323794
- Slevinski, S.E., Jr. (2010a). *International SignWriting Alphabet 2010 - HTML Reference*. Retrieved November 10, 2013, from <http://www.signbank.org/iswa>.
- Slevinski, S.E., Jr. (2010b). *SignMaker*. Retrieved August 21, 2014, from <http://www.signbank.org/signpuddle2.0/signmaker.php?ui=1&sgn=63>.
- Srihari, S., Lam, S., Govindaraju, V., Srihari, R., & Hull, J. (1992). *Doc-*



- ument understanding: Research direction* (Tech. Rep.). Center of Excellence For Document Analysis and Recognition.
- Stentiford, F., & Mortimer, R. (1983, Jan). Some new heuristics for thinning binary handprinted characters for OCR. *IEEE Transactions on Systems, Man and Cybernetics*, *SMC13*(1), 81–84. doi: 10.1109/TSMC.1983.6313034
- Stockman, G., & Shapiro, L. (2001). *Computer Vision* (1st ed.). Upper Saddle River, NJ: Prentice Hall.
- Stokoe, W. (1960). Sign Language structure: an outline of the visual communication systems of the American deaf. *Studies in Linguistics*, *8*, occasional papers.
- Sukhaswami, R., Seetharamulu, P., & Pujari, A. (1995). Recognition of Telugu characters using Neural networks. *International Journal of Neural Systems*, *6*, 317–357.
- Sutton, V. (n.d.). *ASL Wikipedia Project*. Retrieved November 10, 2013, from [http://ase.wikipedia.wmflabs.org/wiki/Main\\_Page](http://ase.wikipedia.wmflabs.org/wiki/Main_Page).
- Sutton, V. (1977). Sutton Movement Shorthand: Writing Tool for Research. In W. C. Stokoe (Ed.), *Proceedings of the First National Symposium on Sign Language Research & Teaching* (pp. 267–296). Chicago, IL: Department of Health, Education and Welfare.
- Sutton, V. (1983). *A Collection of Classical Ballet Variations*. La Jolla, CA: Center for Sutton Movement Writing.
- Sutton, V. (1993). *SignWriter-At-A-Glance Instruction Manual, SignWriter Computer Program Notebook*. La Jolla, CA: Deaf Action Committee for SignWriting.
- Sutton, V. (1995). *Lessons in SignWriting*. La Jolla, CA: Deaf Action Committee for SignWriting.
- Sutton, V. (1996). *SignWriting For Sign Languages*. Retrieved November 13, 2013, from <http://www.signwriting.org/>.
- Sutton, V., & von der Lieth, L. (1976). *Examples of Notation of the Danish Deaf Sign Language: Notation Samples of Simple Sentences in the Danish Deaf Sign Language and the Danish Mouth-Hand System*. Irvine, CA: Movement Shorthand Society Press.
- Taubman, G. (2005). *Musichand: A handwritten music recognition system* (Tech. Rep.).
- Vanderheiden, G., Chisholm, W., & Jacobs, I. (Eds.). (1999). *Web Content Accessibility Guidelines 1.0*. Retrieved November 11, 2013, from <http://www.w3.org/TR/WCAG10>.
- Wakahara, T., Murase, H., & Odaka, K. (1992). On-line handwritten recognition. In *Proceedings of the IEEE* (Vol. 80, p. 1181–1194).
- Wang, W. (1998). Binary Image Segmentation of Aggregates Based on Polygonal Approximation and Classification of Concavities. *Pattern Recognition*, *31*(10), 1503–1524. doi: [http://dx.doi.org/10.1016/S0031-3203\(97\)00145-3](http://dx.doi.org/10.1016/S0031-3203(97)00145-3)
- Wharton, C., Rieman, J., Lewis, C., & Polson, P. (1994). Usability Inspection Methods. In J. Nielsen & R. L. Mack (Eds.), (pp. 105–140). New

- York, NY: John Wiley & Sons, Inc.
- Wixon, D., Holtzblatt, K., & Knox, S. (1990). Contextual design: an emergent view of system design. In J. Carrasco & W. J. (Eds.), *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people* (pp. 329–336). New York, NY: ACM.
- World Health Organization. (2012). *Deafness and hearing impairment* (Tech. Rep.). Retrieved November 15, 2013, from [www.who.int/mediacentre/factsheets/fs300/en/index.html](http://www.who.int/mediacentre/factsheets/fs300/en/index.html).
- Zhang, G. (2000). Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 30(4), 451–462. doi: 10.1109/5326.897072
- Zhang, T., & Suen, C. (1984). A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3), 236–239.

AUTHOR: Fabrizio BORGIA

TITLE: Informatisation of a graphic form of Sign Languages: application to SignWriting

THESIS SUPERVISORS: Michel DAYDE, Maria DE MARSICO

PLACE AND DATE OF DEFENCE: Rome, 30/03/2015

The studies and the software presented in this work are addressed to a relevant minority of our society, namely deaf people. Many studies demonstrate that, for several reasons, deaf people experience significant difficulties in exploiting a Vocal Language (VL English, Chinese, etc.). In fact, many of them prefer to communicate using Sign Language (SL). As computer scientists, we observed that SLs are currently a set of underrepresented linguistic minorities in the digital world. As a matter of fact, deaf people are among those individuals which are mostly affected by the digital divide. This work is our contribution towards levelling the digital divide affecting deaf people. In particular, we focused on the computer handling of SignWriting, which is one of the most promising systems devised to write SLs.

OPTICAL CHARACTER RECOGNITION

SIGN LANGUAGE

HUMAN-COMPUTER INTERACTION

ACCESSIBILITY

IMAGE PROCESSING

DEAF

USER INTERFACES

SIGNWRITING

DISCIPLINE: Computer Science

**AUTEUR:** Fabrizio BORGIA

**TITRE:** Informatisation d'une forme graphique des Langues des Signes: application au système d'écriture SignWriting

**DIRECTEURS DE THÈSE:** Michel DAYDE, Maria DE MARSICO

**LIEU ET DATE DE SOUTENANCE:** Rome, 30/03/2015

---

Les recherches et les logiciels présentés dans cette étude s'adressent à une importante minorité au sein de notre société, à savoir la communauté des sourdes. De nombreuses recherches démontrent que les sourdes se heurtent à de grosses difficultés avec la langue vocale, ce qui explique pourquoi la plupart d'entre eux préfère communiquer dans la langue des signes. Du point de vue des sciences de l'information, les LS constituent un groupe de minorités linguistiques peu représentées dans l'univers du numérique. Et, de fait, les sourds sont les sujets les plus touchés par la fracture numérique. Cette étude veut donc être une contribution pour tenter de resserrer ce fracture numérique qui pénalise les sourdes. Pour ce faire, nous nous sommes principalement concentrés sur l'informatisation de SignWriting, qui constitue l'un des systèmes les plus prometteurs pour écrire la LS.

---

**OPTICAL CHARACTER RECOGNITION  
SIGN LANGUAGE  
HUMAN-COMPUTER INTERACTION  
ACCESSIBILITY  
IMAGE PROCESSING  
DEAF  
USER INTERFACES  
SIGNWRITING**

---

**DISCIPLINE:** Informatique

---

Institut de Recherche en Informatique de Toulouse  
Université Toulouse 3 Paul Sabatier  
118 Route de Narbonne  
F-31062 TOULOUSE CEDEX 9