Technische Universität München

Fakultät für Mathematik
Lehrstuhl für Geometrie und Visualisierung

Stroke-based Handwriting Recognition: Theory and Applications

Bernhard Odin Werner

Vollständiger Abdruck der von der Fakultät für Mathematik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Prof. Dr. Massimo Fornasier
Prüfer der Dissertation:	1.	Prof. Dr. Dr. Jürgen Richter-Gebert
	2.	Prof. Dr. Kristina Reiss
	3.	Prof. Dr. Sven de Vries Universität Trier

Die Dissertation wurde am 12.02.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Mathematik am 16.07.2019 angenommen.

Acknowledgements

First and foremost I want to thank Jürgen Richter-Gebert and Kristina Reiss for this amazing opportunity to work in two different scientific fields, and Frank Reinhold and Stefan Hoch for the marvellous collaboration. I am very grateful that I learned many different things unrelated to my actual dissertation and happy that we managed to produce a piece of software that met the appeal of students, teachers and other researchers alike. The experiences I made during ALICE will, without a doubt, shape my professional future.

I want to thank everyone at the chairs for Geometry and Visualisation, for Algebra, and for Mathematics Education for all the enlightening academic discussions and the occasional diverting private ones. I want to thank all my friends in Traunstein, Munich, Münster, Berlin and Boston for many, many diverting private discussions and the occasional academic ones.

I want to thank Matt, Noel, Amora, Pedro, Lena, Kevin and Madeline for showing me how to climb a mountain; and Steve and every other Rebel for your support and help on my epic and less epic quests over the last four years.

Last, but most important, I want to thank Jan-Christian and Dorothea for keeping me sane during the last few years.

Contents

1 Writing on touchscreens in technology, education & mathematics

"What are letters?"
"Kinda like mediaglyphics except they're all black, and they're tiny, they don't move, they're old and boring and really hard to read. But you can use 'em to make short words for long words."

— Harv to Nell, *The Diamond Age*, by Neal Stephenson

A very well-known problem in machine learning and pattern recognition is to identify written/drawn characters, icons and figures. This usually comes in two flavours: off-line and on-line. The former deals with pixel data as the raw input and encompasses scanned book pages, handwritten exams by students, and even blackboards seen in video-recorded lessons. The latter, however, may use the precise position of the tip of a pen or stylus sampled by a finite number of points and is used when writing on an electronic whiteboard or a touchscreen. In this thesis, we will focus on the latter one and explore several approaches to this task.

When surveying the literature on on-line handwriting recognition, three things become eminent: First, as a purely practical problem, handwriting recognition is "solved". There are more than enough different algorithms that work good enough. Second, the various approaches to handwriting recognition (HWR for short) differ greatly depending on the area of application — Latin letters, simple geometric shapes, Chinese characters, etc. Third, most development processes for these algorithms seem to start with the application in mind and are tailored to recognise a specific symbol set.

This thesis aims to shift the thought process away from the purely pragmatic mindset and to establish mathematical foundations that can be applied to various tasks with only small adaptations. Additionally we want to build the classification process in a way that produces descriptive, humanly understandable results. Both these goals are motivated by the use in interactive educational software.

The practical background for this thesis is given by the *ALICE:fractions* project which was established in 2015 by the Heinz Nixdorf Foundation. In it, the Heinz Nixdorf chair for Mathematics Education and the chair for Geometry and Visualization at the TUM together created an interactive schoolbook on iPads for sixth-graders on the topic of fractions. The handwriting recognition algorithm developed for it is the guidepost for the theoretical considerations here.

So, in order to motivate the assumptions and requirements for the algorithm, we will also discuss the role of handwriting recognition in educational software and describe how it was implemented in ALICE:fractions. Before we do so, we start with an overview of the history of technical devices capable of handwriting recognition and how they benefit from such programs.

1.1 Handwriting recognition in technology

To understand the peculiarities of tablets and other touch surface devices better, we give a brief overview of how this technology arose.

1.1.1 Computers and touch surfaces

When asked about "computers on which you can write on the screen" most people nowadays think about state-of-the-art touchscreen devices: smartphones, tablets and video game consoles. People old enough to remember might come up with Personal Digital Assistants (PDAs) like the *PalmPilot*, which have been prominent in the 1990s. That decade saw a surge of other technical gadgets like mobile phones and virtual reality devices.

The image of the 1990s was shaped by these technical advances even though not all of them were successful.[1] Of course, the origins of all these technologies lie farther in the past. For example, mobile phones — in the shape of car phones — were made public in 1946 by *Bell System*.

The advent of interactive computer screens came shortly after screens were added to computers at all. Before the 1960s computers mostly had teleprinters (also known as teletypes) as their "graphical" output device. These were typewriter-like machines that printed the relevant output data on paper. The technology of screens was of course known in 1960, but updating the data to be displayed was very memory-expensive and, therefore, screens were used scarcely. Even after introducing a fixed character set to be displayed — circumventing the need to update every light point individually in the cathode ray tubes — screens were mostly used to show process data. "Proper" output was still printed to paper via teletypes.

Then, in 1962 and 1963, three major inventions saw the light of the world. First, Steve Russell and several other members of the *Tech Model Railroad Club* at MIT created *Spacewar!*, which is seen by many as the first video game in history and which is, therefore, one of the first "interactive" programs. The

[1]The failed *Nintendo Virtual Boy* comes to mind.

significance of this lies in the formal definition of the term "interactive": It is ambiguous, but most attempts to define it are in the vein of:

*An **interactive** software is a computer program which enables and demands input from a (human) user while it is running.*

See, for example, [18]. What that means — especially in the context of early computer science — is that an interactive software is not a glorified calculator that gets started and then works away for hours uninfluenced by any human. It requires frequent, near constant back-and-forth with a user.[2]

Second, Ivan Sutherland invented *Sketchpad* — also at the MIT — as part of his PHD thesis in 1963; which made him the father of interactive computing in many eyes. This program used the input of a light pen: a light-sensitive pen that detects the electron beam that generates the image on screen. Because the beam moves with a known speed over a known path across the screen, the light pen is able to compute the exact pixel it points to. Sketchpad uses this information to allow the user to draw straight lines, circles and other simple geometric shapes "onto" the screen and manipulate them in real time. This is the predecessor of all touch surfaces we are interested in this thesis. Both applications make it necessary to update screens fast and often and made it clear that, from then on, it is highly desired.

Third, Douglas Engelbart invented the computer mouse only a year later. However, for our concerns here, computer mice are less important as an input device. But it and its success made it clear that the interactivity of computer programs will come to the fore, and that a focus will be put on special peripherals for this interaction.

So, even in the early 1960s, everything was set for educational software on a technological level. However, the mouse gained popularity over the light pen over the next decades.[3]

We saw the first big wave of widespread devices with touch surfaces in PDAs

[2]This is also know as 'on-line processing'; in contrast to 'batch processing' which dominated the early years of computer science. Note the double use of the term 'on-line' both in the context of processing and handwriting recognition.

[3]Maybe because holding a pen in front of a screen is much more strenuous than pushing a mouse over a flat surface? This phenomenon is sometimes called 'gorilla arm'.

in the 1980s and 1990s starting with the *Psion Organizer* in 1984 which evolved into smartphones during the first decade of the 21st century. Parallel to this advancement of computers with a touch-sensitive surface or screen, graphics tablets were developed. These are external devices transmitting the movement of a pen or stylus one-to-one to a computer. At the base level, they serve as a simple alternative for a computer mouse or directional keys. But the main application lies in graphics design, art and handwriting input. The first proper graphics tablet for home computers though was *Koala Technologies' KoalaPad* produced from 1983 for several computers including the *Apple II* and the *Commodore 64*. Around the same time, several handwriting recognition programs started to get distributed taking advantage of both touch screens and graphics tablets.

The culmination of all these various technologies — at least seen from the year 2019 — are tablet computers. The idea of book- and notepad-like computers is prevalent throughout the history of computers as can be seen in many works of science fiction as, for example, Isaac Asimov's novel *Foundation* (1951), Arthur C. Clarke and Stanley Kubrick's film *2001: A Space Odyssey* (1968) and Neal Stephenson's novel *The Diamond Age* (1995). However, as such a device has to be small by design, it took a bit longer to see the first models. The earliest predecessor of what nowadays is considered a tablet can be found in 1989: the *GRiDPad*. But the big breakthrough for tablet computer came in 2010 with *Apple*'s *iPad*. Since then, tablets permeated more and more areas of everyday life.

It is obviously hard to predict how the future of computers and, in particular, how in- and output devices will look like. However, as of 2019, there is a push on advancing machine learning based on neural networks. Also, the progress on hardware performance — forecast by Moore's law (see [38]) and similar exponential predictions on technological growth — seem to continue for at least a couple of years. Moreover, virtual reality is on the rise, and there is even slow but steady progress on haptic feedback — as can be seen, for example, in the actuators used in mobile phones (like the *iPhone X*) and video game controllers (like the *Nintendo Joy-Cons*) to generate "meaningful" vibrations, and in the trackpads used on the newer *Apple MacBook* series simulating a click via a small hammer.

This puts the technology of tablets in an interesting spot in the year 2018. On the one hand, such devices are capable of real-time computations (and high-quality rendering). On single-purpose HWR applications — e.g., recognising Latin letters, Chinese characters, simple geometric forms — they perform exceptionally well. And while reducing the test error rate further when it already is well-below one percent[4] is often very hard, it is mostly unnoticeable in practice.

On the other hand, sophisticated machine learning techniques need to be trained and fine-tuned and can often not be adapted on the fly to expand the symbol pool. Even if these algorithms allow such flexibility, asking the user to provide a significant amount of samples for the new characters is impractical. And most mobile devices still lack the computational power to re-train the algorithm; especially when the main application program is supposed to continue.[5]

Moreover, having to develop different variations of symbol recognition for different character sets to be included in the same program takes up many resources. However, having a single recognition system built for all symbols will usually be rather ineffective. See, for example, the *Detexify* project by Daniel Kirsch, [32]: while the algorithm has solid theoretical foundations, it just cannot handle the sheer amount of all LaTeX symbols.[6]

So, despite the rise of neural networks and powerful (soft) artificial intelligence, a more heuristic approach to handwriting recognition is still relevant. And works like [11] show that there is an interest in it. However, to understand why handwriting recognition is at all relevant, especially on tablets, we have to

[4]For example, a drop by 0.04% when comparing the latest two MNIST database entries in [35] in the category 'convolutional nets'.

[5]There is, of course, always the possibility to let both the training and the classification be executed on a dedicated web server. But even in 2019 the wifi coverage in schools and other public buildings is not universal. And relying on electronic devices to be always connected to the internet creates many other problems.

[6]Users of *Detexify* can provide their own samples. So, common, often used symbols are recognised much easier by this program.

look at it from the point of view of a user.

1.1.2 User experience

The rise of interactive software created another concept in its wake: user experience. It is usually defined as

how a user feels and reacts to the use of a program

or similar to that. See [34] and [37] for some alternatives.[7] Some questions that lead to good user experience are whether buttons with similar design have similar functions, whether often-used tools are positioned at a high level in a drop-down menu or whether there is significant input lag using a peripheral. User experience also encompasses the vital topic of accessibility options for disabled people (and others).[8]

Seen from another perspective, user experience tries to facilitate optimal *flow*. This term was coined by psychologist Mihály Csíkszentmihályi, most famously via his book [10], and it describes a mental state in which a person is completely absorbed in a task. With this in mind, improving user experience can be seen as the effort to reduce obstructions during a working process to allow the user to work as efficiently as possible. For programs on tablets, in particular, this means to make essential elements like buttons and (text) input areas large and easily identifiable (cf. [61]). Moreover, buttons should be at the lower side of the screen, and interface elements that display information should be at the top as the user will cover the screen with her hand while interacting with various elements.[9] As a rule of thumb: the easier a program is to operate, the better.

Handwriting recognition now is one significant contribution to immersive work. There are two main reasons for that: Firstly, many programs that utilise the touchscreen need the whole area of the screen. Thus, an on-screen keyboard

[7]There is also an International Standard defining user experience in this vein for all products. However, the definition given here is enough for our considerations.

[8]On a base level, this means a colour palette for colour-blind users or the option to increase the font size.

[9]Designing for left- and right-handed users is another challenge. So, arranging screen elements vertically makes it easier overall.

for text input usually has to pop up whenever needed. As its buttons have to be of a certain size to be usable — preventing the notorious fat-finger problem — the keyboard has to be large and will subsequently cover a significant part of the application interface. Secondly, the ability to recognise hand-written/ -drawn symbols increases the interaction possibilities; especially when control gestures are implemented. This is less of a problem with graphics tablets as they are usually attached to a "proper" computer that also has a physical keyboard. On tablets, however, this becomes relevant as they are more limited as an input device. The basic input methods are a *Home Button* (which does not even exist on the *iPhone X*), volume buttons on the side and a stand-by button (which also falls more and more out of fashion). "Advanced" input factors are, for example, 6-axis accelerometers, gyroscopes and cameras. These are powerful tools that allow the use sophisticated applications. However, they are highly specialised and are rarely used for "menial tasks" like operating an internet browser, a music player or a word processor. This leaves the touchscreen as main interaction method between user and device.

The basic interaction method between a user and the touchscreen is the sequence of Finger Down, Finger Move and Finger Up. Or, in a other words, a stroke with a finger (or stylus). All interaction falls back to these three building blocks: E.g., tapping on a spot on the screen is given by a very short Finger Move phase; and multi-touch gestures employ several of these sequences at the same time. For our goal to analyse handwriting, we assume that these strokes form concrete symbols and characters on the screen. They are characterised by having a concrete, abstract meaning, but not a unique way to write them. We will talk about this dissonance in Section 2.3.

Apart from the fundamental advantages as an interaction method between humans and computers, HWR has much potential in interactive educational software. Flight simulators have shown for years that such educational and training programs can be very successful if they are coupled with the right devices and peripherals. And with the advancement in both hardware and software, there is tremendous potential across all subjects and across many areas of live. Some examples for this that already exist are: virtual tours through ancient Egypt; surgery simulations for young doctors without the danger to harm humans (or animals); teaching fractions in 6th grade by cutting and distributing virtual pizzas on an iPad. The primary reason why these applications work is their potential for immersion. But it is only fulfilled if working with interactive software is not hindered by clunky designs (of hardware or software).

Looking specifically at interactive educational software for schools it is evident that tablets are a very promising tool. From a design point of view, this becomes clear as tablets are basically electronic books and they built around that idea. But even though digital tools, in general, are used for a long time now in schools (see for example [65], [54] or [57]), there are not many investigations into tablets specifically (cf. [21]). And as most studies in this area are devised to have specific environments and set-ups, it is understandable to look at meta-analyses for deeper insights into the feasibility of tablets in schools (cf. [20] or [21]).

Specifically for handwriting input there are, for example, the works by Aziz et al. (see [2]) and by the group around Read and MacFarlane (see [43] and [39]) who study the design of HWR software to teach preschool and primary school children. In particular, as [44] and [42] show, there are little problems with children using a touchscreen to write per se — especially compared to an ordinary keyboard. However, many, many small issues hinder the user experience. Notably:

- The slow writing speed of children that results in wrong parsing and grouping of strokes into words. (See Section 5.1.1 in [44].)
- Children trying to "fix" letters by adding more strokes and lines; regularly after they have continued to write the next letters. (See Section 5.2.3 in [44].)

The last point is especially interesting as this exemplifies the dissonance between off- and on-line handwriting recognition we already mentioned at the very start: The former focusses more on the images produces while the latter concentrates on the movement of the pen/finger. When using this second, gesture-focussed approach, adding additional lines is detrimental to the recognition process and has to be taken into account separately.

Due to the relatively small number of research endeavours into the feasibility of tablets and the fact that it is approached from an academic angle, a flaw emerges that can be seen, for example, in the handwriting software used in [44] and [39]: The software itself is rarely optimised for broad use and often lacks polish.

In the next section, we will present the ALICE study which motivated this thesis. It takes full advantage of iPads as a digital learning tool. Moreover, the project was built with both an academic and a practical goal in mind: the two intervention studies conducted explore how effective interactive educational software can be; at least for the topic of fractions. Also, the iBook[10] created in the process was not only designed for the studies but to be a standalone, ready-made product for school lessons.

This was achieved by a diverse team of mathematicians, teachers and programmers[11] and a solid footing in Educational Science. In particular, the two main theories ALICE is based on — Cognitive Load Theory and Embodied Cognition — are closely related to *flow* and optimal user experience. We will summarise these theories, but only to the extent of understanding their role in the design of ALICE.

As a last note before talking about ALICE in detail: There may be many more applications for the methods presented here and in other HWR research than just the actual recognition of what was written. Apart from obvious ideas like signature identification, there are even utilisations in medicine like diagnosing Parkinson's disease (see [73]). However, here we will proceed with the use as an input method in interactive educational software in mind.

[10]Apple re-branded the program *iBook* as *Apple Book*. We will use the term *iBook* throughout this thesis to denote the file format that can be read via this program on Apple devices.

[11]Every member fulfilled at least two of these roles.

1.2 The ALICE project

As discussed in the previous section, computer hard- and software changed and grew rapidly in the second half of the 20th century. Moreover, children are nowadays exposed to technical devices like smartphones from a very young age on. Their growing brains allow them to learn many complicated things very fast and, most importantly, in a very natural way. So natural in fact that we have a word for them: Digital natives. It was coined by Marc Prensky in his 2001 paper *Digital Natives, Digital Immigrants* [41], but it can even be traced back to John Barlow's *A Declaration of the Independence of Cyberspace* [4] from 1996.

This contrast between digital natives and digital immigrants — people who grew up with computer and people who learned to work with them later in their lives, respectively — created a substantial problem in the educational system: While most parents, teachers and government official agree that working with the new technology and media is essential and that we should take the chance to incorporate them in classrooms and curricula, there is no clear plan on how to do this. Two main hurdles are the training of the teachers and the development of adequate software.[12] Since many teachers, as well as the authors of school books and employees of publishing houses, are not intuitively familiar with tablets (yet), the training becomes time-consuming. Moreover, early software lacks the utilisation of computers in general and touchscreens in particular. Notably, many digital schoolbooks are still just scans of ordinary schoolbooks and do not take digital, interactive media to their full potential.

Here we will now present the research project ALICE:fractions which explores the theoretical foundations of sensible interactive educational software design and the appurtenant iBook implementing these ideas. This project tackles the second of the problems mentioned above directly: How can/should educational software for tablets look like and function? However, it also brushes on the first one by taking the feedback of both teachers and pupils

[12]Aside from obvious organisational problems like buying computers in class sets.

into account. The easier such a program is to use, the fewer resources have to be invested to train the teachers, and the fewer reservations exist to use it.

Thematically it is restricted to fractions — in particular to introducing their fundamental properties. The primary reason for this is that this topic lends itself very well to an enactive approach. Also, it is an area where the performance of students leaves a lot to be desired, so, there is room for measurable improvements.

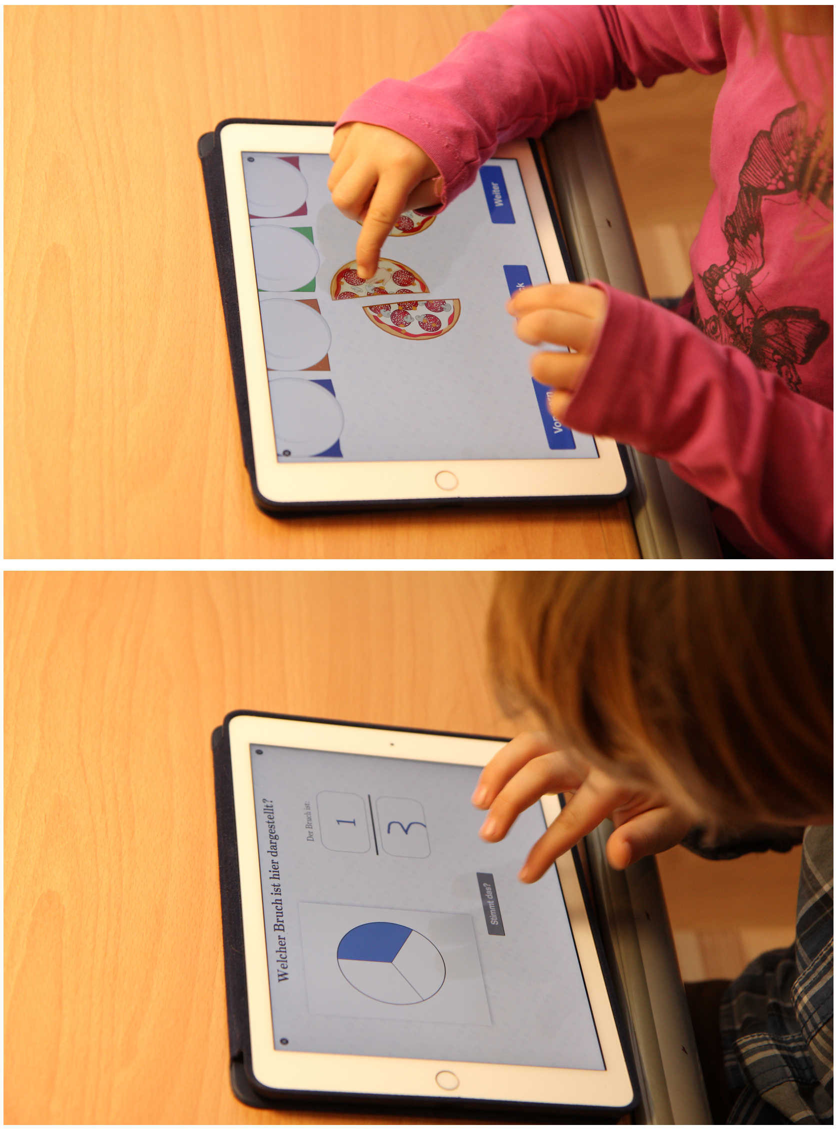


Figure 1.1: *Children working with ALICE:fractions.*

1.2.1 Overview & design

The project ALICE:fractions was created under the (German) name

Lernen mit dem Tablet-PC: Eine Einführung in das Bruchrechnen für Klasse 6,

which roughly translates to

Learning with tablets PCs: An introduction to fractions for grade 6.

As a joint research project between the Heinz Nixdorf Chair for Mathematics Education and the chair for Geometry and Visualization and funded by the Heinz Nixdorf Foundation, it had the goal to develop an interactive iBook on iPads to teach fractions. Three PhD students worked on this project under the supervision of Prof. Dr. Kristina Reiss and Prof. Dr. Dr. Jürgen Richter-Gebert. These students — Frank Reinhold, Stefan Hoch and the author — as well as the two project leads have backgrounds across mathematics, education and software development. Taking advantage of this, the entire content of the resulting iBook was created from the ground up. In particular, the interactive exercises were built using the dynamic geometry software *CindyJS* (see [13]).

The main design goal was to take advantage of the computer-environment to react in real time to the behaviour and answers of the students, and the desired outcome was to foster an intuitive and descriptive understanding of fractions. To emphasise these two aspects, the name of the project was changed to ALICE which is an acronym for

__A__daptive __L__earning in an __I__nteractive __C__omputer-supported __E__nvironment.

The addendum *fractions* shall accentuate that this general design idea is independent of the actual topic or subject and the hope to apply the results to other areas.[13]

Here in this section, we will give a brief overview of the project — explaining the importance of the adaptivity and interactivity as well as why the topic of fractions is both relevant and interesting in this context. We will also explain

[13]Topics from geometry and stochastics are especially desirable. They lend themselves well to an interactive learning experience. And various professors at the TUM spoke out their interest in seeing them come to fruition.

the role of handwriting recognition in this project and how it fits the design of the iBook. This will be a summary of ALICE:fractions in order to provide the practical framework for the work presented here in this thesis. For more detailed information, please see the other two (German) dissertations written within this project:

— *Mathematikdidaktische und psychologische Perspektiven zur Wirksamkeit von Tablet-PCs bei der Entwicklung des Bruchzahlbegriffs – Eine empirische Studie in Jahrgangsstufe 6* by Frank Reinhold describing the theoretical background of the iBook design and analysing how students improved using it. See [45].

— *Prozessdaten aus digitalen Schulbüchern* (working title) by Stefan Hoch analysing the user behaviour and inferring how successful pupils work. See [22].

The two main theories from educational science used in ALICE:fractions are Cognitive Load and Embodied Cognition. These two concepts have been developed in the late 20th century and gained much more significance since the introduction of tablet computers into schools. Here we will give a brief overview of them and describe how they influenced the development of both the ALICE iBook in general and the handwriting recognition software in particular.

The basic idea behind the Cognitive Load Theory (see the works of Sweller starting in 1988 with [62]) is that any form of mental activity uses a form of cognitive capacity and that this capacity is finite. I.e., the more it is used throughout the day, the more laborious mental tasks become. There are many different experiments from psychology illustrating this and similar effects.[14] In the context of education, the core insight of Cognitive Load Theory is that there are internal and extraneous factors contributing to how taxing and hard to solve

[14]One interesting fact, for example, is how every form of mental strain can affect every other mental effort. This is illustrated by a famous experiment from Baumeister, Bratslavsky, Muraven and Tice from 1998, see [5]: Participants waited in a room with two plates of cookies and radishes, respectively. One group was told that they have to eat the radishes while the other group could do/eat whatever they wanted. In a relative demanding puzzle afterwards, people from group one performed significantly worse than the ones in group two.

an exercise is. Moreover, these factor stands in the way of the actual learning process. (Cf. [63].)

Intrinsic factors are the parts that constitute the core problem of solving an exercise. E.g., doing mathematical computations is hard in itself, and solving $17 \cdot 39$ is much harder than solving $2 \cdot 2$. (Of course, barely anyone “solves” $2 \cdot 2$; it is usually recalled from memory.) The relevance of considering these intrinsic factors in designing educational environments is, among other things, whether solving many, relatively easy exercises is more beneficial than solving fewer, hard exercises.

Extraneous factors are the elements outside the actual task that contribute to its difficulty nevertheless. They include things like the presence/absence and quality of a sketch, the font style, colour, size and length of the instructional text as well as the complexity of the words used. However, there are also external factors in the environment like loud (unpleasant) noises and bad lighting.

There is the third category of germane cognitive load which describes the work someone puts into solving a problem by creating and using mental schemata. This is the desired focus of mental work while learning. I.e., the primary goal of designing exercises, explanatory texts, diagrams, etc. is to reduce intrinsic and extraneous factors in order to facilitate this process.

The second building block of ALICE is Embodied Cognition. Its idea is that performing appropriate actions with one's hands (or body) accompanying mental work can improve the degree and speed of understanding (and that performing inapt actions hinders it). An exact definition of embodied cognition is amiguous[15]; the design of ALICE revolves around **simple embodiment** as introduced by Clark on page 348 of [9]:

[Simple embodiment] concentrates attention on an inner representational resource ... and is exploring the ways in which usefulness in the guidance of real-world action can both constrain and inform the nature of inner representations and processing.

We will return to Embodied Cognition after quickly presenting why the topic of fractions was chosen to be the focus of ALICE.

[15]See[45] for references to eight different versions.

Studies like [3] show that there are connections between an understanding of fractions and more complex topics.[16] And the extent of the inability to work efficiently with fractions is not only limited to schoolchildren — it can also be found in adults. Moreover, there are various psychological effects which are at least partially explicable by this lack of an intuitive understanding of fractions: For example, Kimihiko Yamagishi has shown in [70] that a fictitious disease with a mortality rate of 1286 persons out of 10000 is perceived as worse than another one with a mortality rate of 24.14 persons out of 100.

The reason why fractions are so hard to grasp is an aggregation and combination of many different conceptual changes, which emerge when going from integers to rational numbers, and the subsequent mishandling of fractions.[17] Concepts that have to change when learning fractions are, for example, the idea of a unique successor for rational numbers, that a single number always has a unique representation and that multiplication makes numbers bigger. Additionally, fractions fulfil many different roles: as ratios, quotients, operators, portions, etc. And all of these are different conceptions in the mind of children. (Cf. [40].) See [45] for a detailed discussion on how and why fractions are complicated to understand when first introduced in grade 6.

Fractions and fractional calculus are decidedly descriptive and geometric concepts. Introducing them as a notation without adequately explaining what the names 'numerator' and 'denominator' mean leads to pupils seeing them as a pair of "normal" numbers without any significant relation. And then the rules for expanding/reducing, comparing, addition and multiplication have no reference to real objects and are perceived as random.

Here comes Embodied Cognition into play: a central focus of the iBook design are the interactive widgets — both for exploratory tasks at the start of each chapter and for ordinary exercises — which predominantly show fractions in iconic representations. Manipulating them "directly" with a finger movement means that abstract operations and concepts — like expanding/reducing

[16]Of course, this is not entirely surprising. Many "higher" topics like solving linear equations and definining the derivative of a function use the knowledge of fractions directly.

[17]The study by Yamagishi illustrates one of these mistakes: denominator neglect. (A term attributed to Paul Slovic.) People often compare fractions by merely comparing the numerators.

or comparing — are linked to "real" ones. The hope behind this design is that the same graphical depictions and manipulations come to mind when the pupils have to solve more classical arithmetic tasks. (And the results of the studies conducted suggest that this is the case.)

With all these ideas in mind, the iBook designed and built during the ALICE project took a certain form. First, all explanations of new concepts are opened by an exploratory task using iconic representations of fractions. Second, a majority of the interactive exercises also make use of iconic representations and allow the pupils to interact as directly as possible with fractions.

The goal to make as many things in the iBook "touchable" now indicates the use of a handwriting recognition algorithm.

1.2.2 ALICE:HWR

As mentioned above in Section 1.1.2, being able to use handwritten words, sketches and gestures directly as input increases the user experience of software that focuses on art and text processing. It makes the touchscreen feel more diverse than it actually is.[18] And in the ALICE iBook, we focus on manual manipulation of "real" and iconic depictions of fractions. Moreover, a work-in-progress feature for a future version of the iBook is a scribble mode. With it, a user can superimpose a semi-transparent layer on which they can write and draw with a small assortment of (digital) pens. As already alluded to in Section 1.1.2, it would break the immersion if an on-screen keyboard popped up whenever the input of numbers is requested.

In tasks that require the pupils to enter running text answers, such a keyboard is unavoidable. However, they are scarce and mostly serve to record pupils thought and allow for a discussion with the teacher afterwards.

Enhancing the benefits of iconic fraction representations via handwriting recognition becomes much more perceptible when sketch recognition is included. This was contemplated during the development of some interactive widgets.

[18]For a similar reason the buttons in the ALICE iBook are drawn and animated in a way that mimics three-dimensionality, and in pizza-cutting tasks a hand or knife is shown at the finger-/pen-down point on the screen.

However, it was not included due to the ambiguity of fraction representations and the problem of evaluating them. In Figure 1.2 we see examples of both these problems.

On top is a typical error by someone who just learned about fractions. When the task is to subdivide a circle into three equal parts, we will often see this picture: A diameter (most often the horizontal one) is divided into three equal parts, and then perpendicular lines to it are drawn to divide the circle. Of course, this is not correct. However, the correct solution of subdividing a circle by two parallel lines is so close to this wrong one that it is nigh impossible to assert the falseness of such a solution with high confidence. At least, when the user cannot provide explanations for their answer. An alternative is to label this kind of solution as wrong, regardless of the position of the parallel lines, and only permit lines through the centre of the circle. But, of course, it is very bad practice to condemn creative solutions just because they do not fit the established mould.

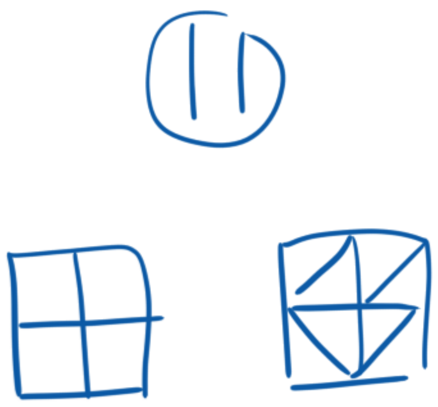


Figure 1.2: *A typical error in subdividing a circle (top) and unusual graphical expanding by* 2 *(bottom).*

The second problem can be observed at the bottom of Figure 1.2. A common way to explain expanding and reducing fractions is via refining and coarsening of an iconic depictions of fraction. As Widget/Exercise 39 in the ALICE iBook shows, this can be done for reducing in a computer environment. However, it is hard to implement intuitive controls for such an exercise and to accurately classify mistakes. In a potential expanding widget we would have to decide which input by the pupils is considered correct. Is every correct subdivision of each rectangle eligible? If not, should the lines that can be added limited? If yes, how can we assure that the student did not find the correct solution by guessing? Also, if the answer options are limited, is there even a benefit in doing this exercise? And judging the correctness of a refinement becomes even more complicated when the pupils use non-straight lines.

These two small examples illustrate that automatic recognition of written and drawn symbols and shapes, while useful in general, is not applicable in every situation — in particular when the interpretation of the input is unclear. Therefore, it is used solely for numbers in the ALICE iBook, and we want to keep that as the prime example/application in mind. Practically, they are limited to the range `1,...,99` as the iBook is aimed at six graders and focusses on introducing fractions as a new concept. But theoretically, any integer is possible.

As a guideline for the design of the HWR algorithm used in ALICE — which we will call ALICE:HWR from now on — we present some criteria that were decided upon at the beginning of the project.

The algorithm should not depend on an extensive training dataset because of two reasons: Firstly, children are notorious for their bad handwriting, and a sub-goal was to foster the proper ways of writing numbers. And since ALICE:HWR is a stroke-based, on-line classifier, it focuses more on the overall movement of the finger or pen tip than the picture created on the touchscreen. So, this sub-goal is compatible with the general approach of the software.

Secondly, it became clear in the early stages of ALICE that we want to "tell" the algorithm how certain strokes look like instead of just "showing" it many samples. This way, it is easier to generalise the algorithm and adapt it to other characters set.

Speaking of generalisation: with a future application of the basic ideas of ALICE to other areas of mathematics or even other subjects in mind, the handwriting recognition algorithm is built universally. It should be easily adaptable to (block) letters, simple geometric forms, symbols from electrical engineering and also control gestures. Because of that, the actual classifying step in

ALICE:HWR is very simple and uses, decidedly, no advanced methods from machine learning.

We revisit these design criteria in Section 1.3. And the actual classification steps of ALICE:HWR will be presented in Section 2.3, Chapter 4 and Section 6.2.

1.2.3 Results of the studies

As as last part of this overview of ALICE:fractions we will give a summary of the studies conducted to test the theoretical hypotheses and practical implementations. We will only give the main results here. A complete overview can be gained from the other two dissertations associated with the project [45] and [22] as well as the following articles:

— General information and overviews can be found in [23], [31], [46], [48], [49], [50] and [53] .

— Analyses of process data like finger movements, time on task and differences in visualisation in circle and bar diagrams are presented in [24], [28] and [30].

— The effect of explaining different strategies to compare fractions is explained in [47] and [52].

— The influence of the demographic factors gender and school type is presented in [29] and [51].

Now, some details about the two studies that were conducted. The first in 2016 at German *Gymnasien* (higher-level high schools) and the second in 2017 at German *Mittelschulen* (lower-level high schools). Both studies had the same set-up. In particular, the participating classes were divided into three groups:

1. A first intervention group in which teachers and pupils worked with the iBook.
2. A second intervention group in which teachers and pupils worked with a printed workbook version of the iBook.
3. A control group in which teachers worked as they always did.

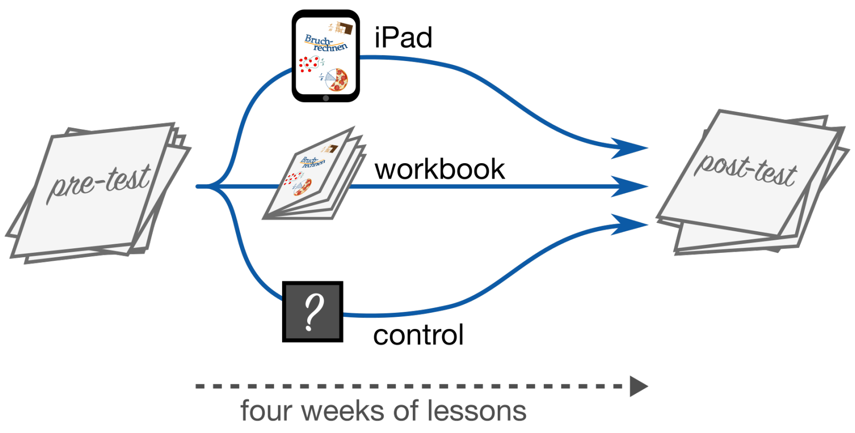


Figure 1.3: *The structure of the* ALICE *studies.*

The original German iBook, an English version of it and the German printed workbook can be found online; see [25], [27] and [26], respectively. The introduction of the second intervention group aimed at separating the effects of the general design of the teaching material and additional benefits of the tablet-based environment. Layout and design of the printed workbook were identical to the iBook. But for interactive exercises, only a finite number of subtasks was printed. Notable characteristics of both studies were the following:

— The scope of the ALICE iBook is the introduction of fraction It covers the following areas of comprehension: understanding fractions both as parts of a whole and as parts of many; expanding and reducing; the number line; mixed fractions and comparing fractions by size. The arithmetic of fractions was omitted.

First, because of time-constrains on the development phases: producing the remaining material would have taken more than a full year if it were to be of the same quality as the introductory part. So, the team decided on improving this introduction based on the results of the first study and to conduct another one at a different school type.

Second, because the introduction can rely more heavily on iconic and graphical depictions — whereas the communication of "complicated" operations like the division of fractions cannot avoid imposing certain abstract rules.

— The layout and design of the iBook are the same as of a regular schoolbook. In particular, it contains motivations, explanations, take-home messages as well as exercises. It can be used together with reading assignments, group work and all other teaching methods.

— The teachers in the first intervention group were asked to work as much as possible with the iBook. However, they were encouraged to use additional material if desired.

— The pupils were tested before and after the intervention. The pre-test was conducted to correct the post-test score for prior knowledge of the children — which they might have gained from being taught at home or having had to repeat the sixth grade. In order to guarantee the reliability of the post-test, the teachers were told the areas of skill that have to be covered. Beyond that, they were given plenty of rope to arrange the content of the lessons.

— The iPads for the study were provided by the Technical University of Munich. Because of that, the pupils were not allowed to take them home. This especially is a point of enquiry for future studies, since using interactive educational software for homework and repetition should have a much bigger effect on the level of comprehension and skill than just using them at school.

In total, 1108 pupils participated in the study. Unfortunately, some were sick during either the pre- or post-test and some did not get permission from their parents to take these tests. Moreover, several classes exceeded the scheduled 15 lessons and had to be excluded from the evaluation.

— 29 classes with a total of 808 children participated in the first study at *Gymnasien*, but only the results of 476 were evaluated. Of these, 156 pupils were in intervention group one, 182 in group two and 138 in the control group.

— 16 classes with a total of 300 children participated in the first study at *Mittelschulen*, but only the results of 236 were evaluated. Of these, 105 pupils were in intervention group one, 64 in group two and 67 in the control group.

The post-tests of both studies tested the knowledge of the pupils in the areas of comprehension mentioned above. Moreover, every test item was categorised in one of three areas of skill:

— Working with and understanding of graphical representations.

— Applying arithmetic rules.

— Explaining mathematical statements and situations.

That means that the task

Draw $\frac{3}{4}$ *of* 4 *pizzas*

falls into the first category, while

Compute $\frac{3}{4} \cdot 4$

falls into the second and

Explain why the following picture shows $\frac{3}{4}$ *of* 4 *pizzas*

into the third. As the iBook focusses much more on the first skill both in conveying new content and in practice, it stands to reason that pupils might be less able to perform arithmetic calculations. Fortunately, this is not the case as our results show. Moreover, only two out of 21 items fell into the third category. They were evaluated only qualitatively for the most part.

The mean solution rates in the post-test split between the first two areas of skill are shown in Figures 1.4 and 1.5. The results of the pre-test at *Mittelschulen* were so low, that the previous knowledge of these pupils can be considered non-existent. These scores are omitted in Figure 1.5.

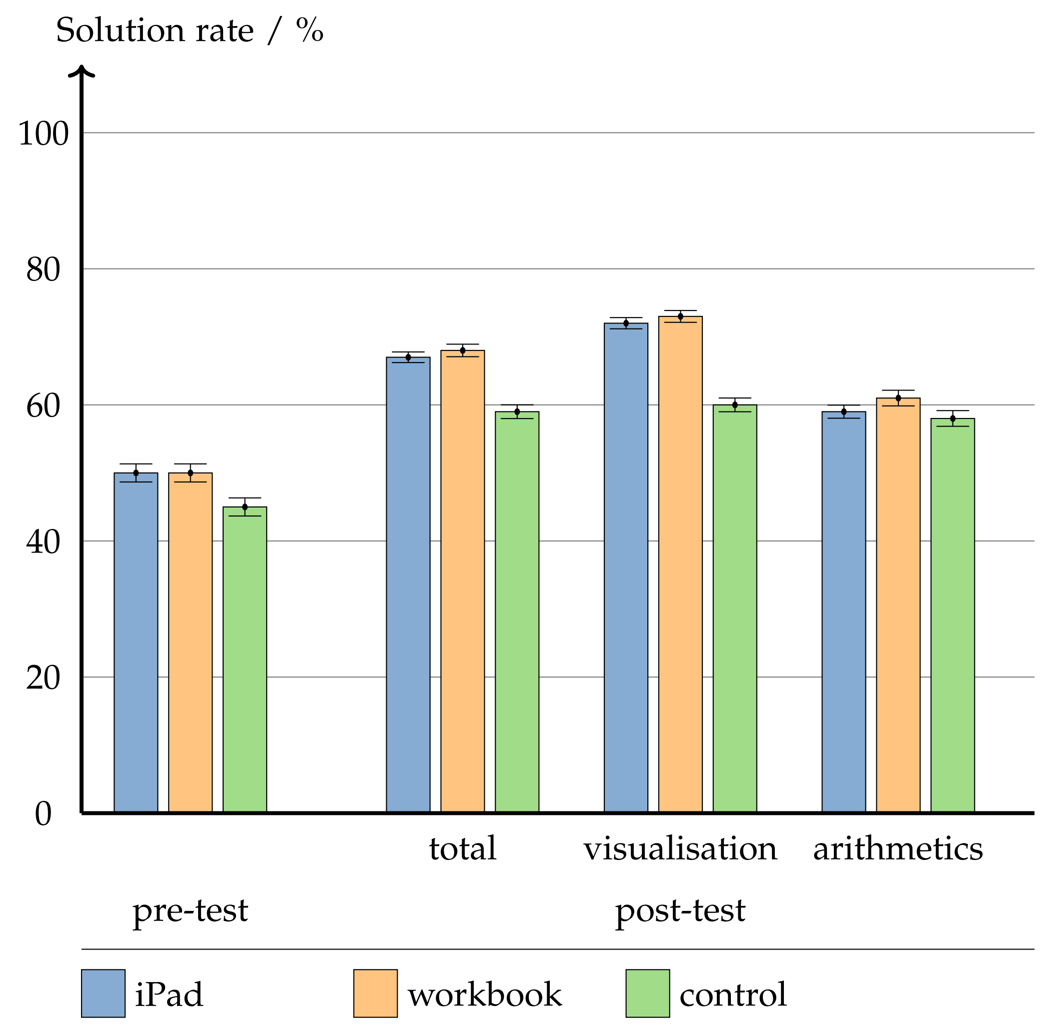


Figure 1.4: *Mean solution rates, together with their 95% confidence intervals, at* Gymnasien.

At *Gymnasien*, both intervention groups were significantly better than the control group. This also holds when we only look at visualisation items. And when evaluating arithmetic items, all three groups performed approximately equal. I.e., even though the iBook focuses on graphical representations, pupils still improved in calculation tasks just as much as "traditionally" taught children.

At *Mittelschulen* there is a much bigger difference between the first intervention group, which worked with iPads, and the others. At the same time, the second intervention group and the control group showed no significant differences.

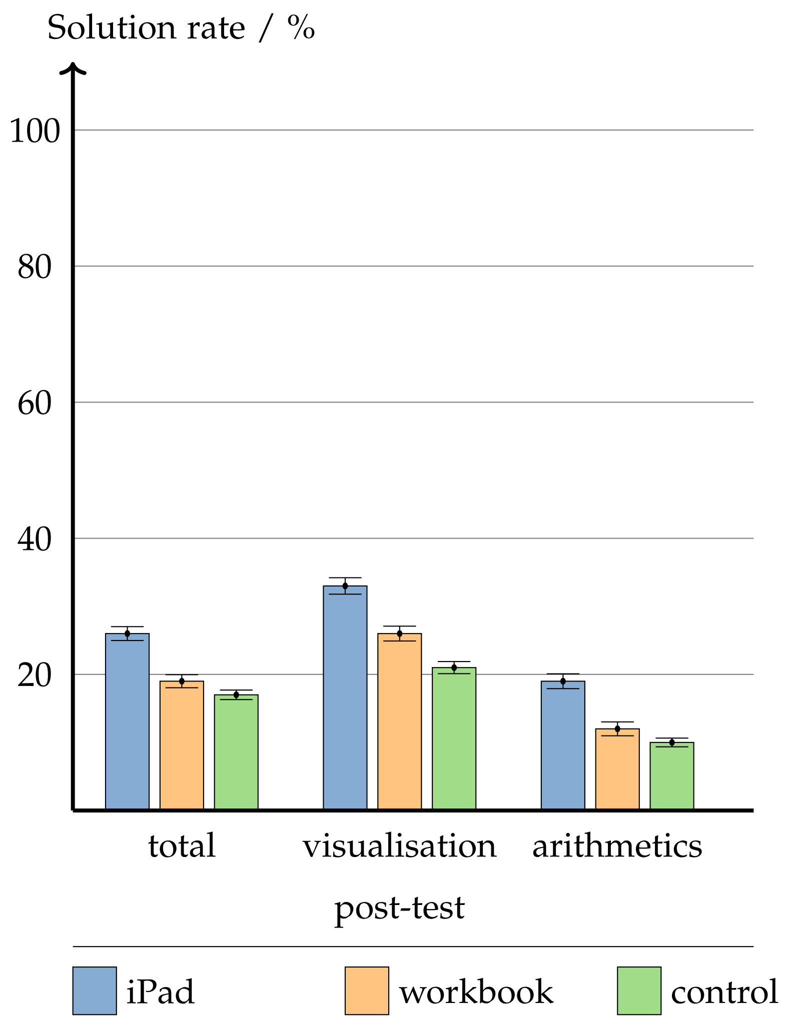


Figure 1.5: *Mean solution rates, together with their* 95% *confidence intervals, at* Mittelschulen.

Overall, an potential explanation for the results might be that for higher-achieving students at *Gymnasien* the interactivity of iPads itself is less important than the overall design and presentation. In contrast to that, children at *Mittelschulen* profited much more from working with iPads.

Apart from these solution rates, much more data was collected and analysed. For example, when comparing the fractions $\frac{5}{8}$ and $\frac{5}{10}$ a large portion of students in both control groups expanded them to have the same denominator. In the intervention groups, however, this strategy was used much less. Another example is the fact that the total time spent on a task is a much better predictor for the performance in a test than the number of individual exercises worked on.

These and other results are discussed in great detail in the aforementioned dissertations and articles.

After this overview we want to shift our attention to actual handwriting recognition. We start by discussing why it is a problem at all.

1.3 The problem of handwriting recognition

When there is a choice, on-line handwriting recognition has the potential to be better simply because it allows to analyse more data: At least the order of the recorded points and at best the exact times at which they were recorded together with additional parameters of the pen like the pressure on the drawing surface or the azimuth of the pen[19].

The rapid advancement in machine learning in the last years, however, makes these differences minuscule. The newest method (as of January 2019) found on the website of the MNIST database of handwritten digits (see [35]) is by Cireşan, Meier and Schmidhuber [8] and they achieve an error rate of 0.23% for the test data set provided by the database. They use deep neural networks which have shown time and time again that they are incredibly in pattern recognition of any kind. The central reservation against the use of neural networks comes from the need to have an extensive training set and to hand-craft the exact architecture of them.[20] The last point, in particular, means that neural networks used for different character sets and areas of application can look vastly different.

In contrast, classical on-line handwriting recognition algorithms are highly heuristic and descriptive. For example, one can decompose a recorded stroke into smaller parts which are relatively easy to recognise as done in [7]. A more common idea is to compute specific geometric properties of a stroke like its length, its local curvature at every point or point spread information like whether more points are in the upper half of the stroke than the lower. Then, after a certain amount of these features are measured they get compared and classified via well-known methods like support vector machines and k-nearest neighbour algorithms.

In order to find or build appropriate features for such an on-line method, we have to talk about the ambiguity of strokes first.

[19]I.e., the angle of the pen when projected onto the drawing surface.

[20]There are, however, some approaches like *NeuroEvolution of Augmented Topologies* to adjust the architecture of a neural network. See [60].

1.3.1 Symbols as abstract pictures

In order to understand how we can describe hand-written symbols, we first illustrate the core problem with this particular data. In chapter 2 of *Understanding Comics – The Invisible Art* (see [36]) Scott McCloud states the problem of abstract symbols the following way:

In the non-pictorial icons, meaning is fixed and absolute. Their appearance doesn't affect their meaning because they represent invisible ideas.

This means that any visual resemblance to an established symbol will make this symbol recognisable. In Figure 1.6 below is a small collection of ways to write the symbol `3` by combining strokes and lines in extremely different ways. In all of them, the number `3` is easy to recognise.[21]

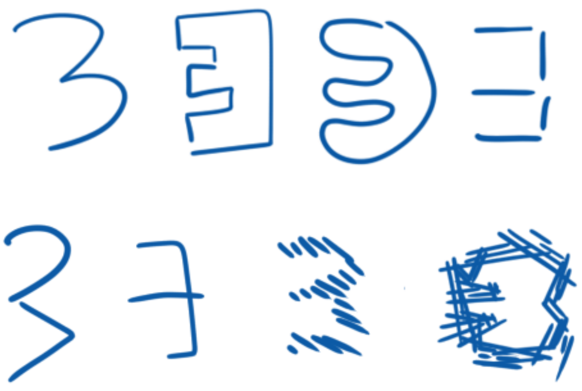


Figure 1.6: *Different iconic representations of the number symbol 3. Inspired by panel 1 on page 28 in* Understanding Comics – The Invisible Art *[36].*

The problem herein is that the inherent `3`-ness of a stroke or a collection of strokes cannot be measured in a deterministic way. Moreover, the semantic meaning of such a picture might depend on the surrounding characters. E.g. the word `test` usually makes more sense than `te5t`. Keeping in mind that we are primarily interested in on-line handwriting recognition used for a school

[21]Of course, this is massively intensified by the fact, that these pictures represent a `3` was given beforehand. Any reader is therefore primed to recognise it.

book, it is clear that the `3` at the top left in Figure 1.6 is what we want to regard as valid.

The goal now is to find geometric and graphic descriptions of what constitutes a stroke like this `3` at the top left. As explained in the sections above, this allows us to better comprehend any subsequent classification process and to control the algorithm more directly. Furthermore, it allows for a potentially automatic explainer: as we aim the application at young students, it is desirable to have future iteration of ALICE:HWR that can explain them directly why something was not recognised.

The procedure to find explicit features, however, is just as ambiguous as the symbols themselves. In particular, we will see that certain properties are very easy to recognise and that — at least for the Arabic numerals — they are often enough to make a decision. But other features will behave more unpredictably.

For example, we will see at various points that recognising straight line segments is very easy and that these particular strokes can be considered the simplest ones. That makes it enticing to create a recognition software for Chinese characters as they quite easily decompose into straight lines. And while [7] illustrates that Fuzzy Logic alone is good enough for the recognition process, there are numerous design challenges for a program used in practice; some of which are explored in [16].

On the other hand, it will turn out that distinguishing between `0` and `8` is relatively hard in contrast. A `0` forms, more or less, a circle and is as such characterised by a constant curvature. The feature to measure this (see Section 2.3.3), however, will be unstable and unreliable due to significant variations both in the way people write `0`'s and `8`'s. So, we combined this feature with others in ALICE:HWR.

However, we will not focus too much on how this problem of unclear strokes and features was tackled in praxis. It is mentioned to some extend in Section 2.3.3 and Chapter 6. The major part of this thesis will deal with more general ideas that can be applied to any form of stroke recognition.

1.3.2 Leading questions

The main source for ALICE:HWR is Delaye and Anquetil's 2013 paper [11] on a broad and universal feature set for on-line handwriting recognition. They showed that the actual classification method used is of lesser importance when based on a good and thorough model of strokes. They surveyed various works on handwriting recognition and compiled 49 features. But no matter how good these are, they are still hand-crafted and cherry-picked. Here in this thesis, we want to approach this with a little bit more generality in mind. The reason for this is a potential transfer of the ideas and results presented here to other areas and projects — e.g. other interactive schoolbooks under the ALICE label or a geometric sketch recognition plug-in for the dynamic geometry software *CindyJS*.

This means that we care less about the actual process of finding features for our symbols set in question and focus on an analysis of the objects we want to classify.

As stated above, we are concerned with on-line handwriting recognition both here in this thesis and in ALICE:HWR. That means that the strokes we record are sequences of points with two coordinates that model the precise finger/pen movement of the user. They form the underlying object for all subsequent analyses and discussions. We will see that the set of all strokes (trivially) forms an affine space. So, the basic geometry of the set of all strokes is very simple. However, we are more interested in the connection between the semantic of a written symbol and its realisation as a specific stroke. So, the fundamental question of this thesis is

(1.) How do strokes of similar shape or meaning relate to each other?

Taking a cue from Felix Klein's Erlangen program, we will focus a large part of this consideration on

(2.) How can strokes be deformed while retaining their meaning?

Finally, we want to keep the requirements of ALICE:HWR in mind. For applications like the ALICE project, it is desirable to have a recognition algorithm that can "tell" the user what went wrong in writing/drawing something. In particular, when the user is a schoolchild and is still learning. This leads to the practical questions

(3.) Can a recognition software be trained by only a small selection of good samples?

(4.) Can the classification itself be done such that its decision making is transparent and explainable?

As mentioned in Section 1.3.1, a large part in answering the last question is to find properties of strokes that have concrete meaning. In this thesis, however, we will, for the most part, assume that this is given and focus a bit more on the classification process itself. Nevertheless, we will always work with concrete and expressive properties.

Question (1.) will permeate every part of this thesis, but will be approached more directly in Chapters 2 and 5. Question (2.) is the central focus of Chapter 3; and that chapter will also answer Question (3.). Finally, Question (4.) will be the topic of Chapter 4.

Before we start, we will describe the structure of this thesis more linearly and fix some general notation.

1.4 Structure & notation

1.4.1 Structure

In this thesis, we make various statements about both the general structure of strokes and features and the way they are used in ALICE:HWR. The goal will always be to explain how and why ALICE:HWR works as well as the more abstract mathematical background. Focus might shift between sections, but both aspects should always be kept in mind. That means, in particular, some statements will be presented more matter-of-factly if they are more concerned with the practical implementation. And some points will be analysed more in-depth if we expect broader usefulness.

Moreover, we will make some implicit assumptions based on the concrete character set we want to classify in ALICE:HWR and the good will of the user. For example:

— Strokes that form Arabic numerals are usually of similar length and therefore we will discretise all of them with the same number of points.

— We assume that the finger/pen will not leave the touchscreen when a single stroke is written. I.e., we assume we do not have to concatenate two or more strokes together before we can analyse them.

— We assume that every number is written the proper way that is taught in German (or rather Bavarian) elementary school. In particular, we will work with only a small set of possible stroke types from which we assemble all numbers.

Some of these points we will repeat when they become relevant. The same points will hold true for all examples throughout this thesis: Some will present very concrete applications, some will be more general.

The order of the chapters will loosely follow the development process of ALICE:HWR. And while Chapter 2 contains the foundation for everything, the others chapters are more or less independent from each other. Their topics are the following:

In Chapter 2 we give a short, self-contained introduction to the three main mathematical theories used to build ALICE:HWR: Projective Geometry, Fuzzy Logic and Formal Concept Analysis. Afterwards we discuss the used base model for strokes and how strokes can be described by features. Lastly, we give an informal overview of how ALICE:HWR works and describe all features it uses.

In Chapter 3 we discuss how strokes can be deformed in different ways. These deformations will all be motivated by concrete visual incentives and, in one way or another, preserve the appearance of the strokes. We will also present applications both for ALICE:HWR and otherwise.

Chapter 4 then contains the three main methods ALICE:HWR uses to actually classify strokes.

In Chapter 5 we introduce a new way to model and describe strokes — in particular their shape. We also show that most transformations introduced in Chapter 3 (which were based on practical considerations) operate in a reasonable way on this new model. Lastly, we discuss how this descriptions allow for similar descriptive analyses as hand-crafted features do.

In Chapter 6 we come back to ALICE:HWR and describe in detail how it uses the ideas from the previous chapter. Moreover, we explain a few points that are essential for the software to work, but which were not introduced beforehand. Also, we give an assessment of the the recognition rate ALICE:HWR achieves.

We close in Chapter 7 with presenting a selection of questions and problems based on the result of this thesis that should be tackled next.

The appendices then contain the manual for the companion iBook [67] for this thesis which shows an interactive demonstration of many concepts presented. Moreover, the complete code for ALICE:HWR version 4 is laid out. This version is the one that is currently (as of January 2019) used in the ALICE iBook [27].

1.4.2 Notation

Throughout this thesis, we will talk about many concepts which have different meanings in our mathematical framework compared to colloquial English. To emphasise this difference, we will use different fonts at three specific occurrences:

First, when we talk about specific symbols written or drawn on a touch surface (and which we want to recognise afterwards), we write their names in typewriter font. E.g. when we talk about the mathematical object

$$C_{p,r} := \{x \in \mathbb{R}^2 \mid \|x - p\|_2 = r\}$$

of all points which have a fixed distance to a fixed point, we will call it a circle. However, when a circle is drawn on a touchscreen, we will call it a `circle`. Likewise, we will denote the number of elements of the set $\{a, b, c, d, e\}$ by 5 or by five; but the S-like symbol representing this number will be called `5` or `five`.

Second, when we talk about specific geometric properties of lines and symbols drawn on a touch surface, we will write their name in small capitals. E.g., we might talk about the length of a vector (i.e. its Euclidean norm) as a mathematical concept or about the LENGTH of a stroke on a touchscreen as one of its specific geometric features.

Third, we will give heuristic rules in italic and centralised text. E.g. we might say that

all digits from `0` to `9` consist of at most two strokes.

Fourth, we write a $\triangle$ at the right side of a page after the end of definitions, remarks and examples in order to clarify where "ordinary" prose starts again.

Fifth, some mathematical notation:

— On $\mathbb{R}$ we denote rounding down to the next integer by $\lfloor . \rfloor$, round up by $\lceil . \rceil$ and "ordinary" rounding by $\lfloor . \rceil$.

— We denote the $n \times n$ unit matrix (over any ring) by I_n.

— We denote the power set of a set X by $\mathcal{P}(X)$.

— When we talk about subsets we will use the symbols $\subset$ and $\subseteq$ interchangeably — both will allow for the subset to be equal to the whole set.

— We abbreviate the zero vector and the all-one vector in $\mathbb{R}^m$ by

$$\mathbf{0} = (0, 0, ..., 0)^T \qquad \text{and} \qquad \mathbf{1} = (1, 1, ..., 1)^T.$$

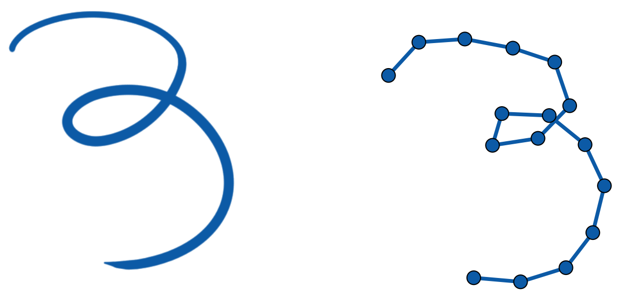


Figure 1.7: *Depicting strokes qualitatively and quantitatively.*

Sixth, when we discuss strokes written on a touchscreen, we will usually do this either qualitatively or quantitatively. In the first case, we are more interested in the image drawn or created on the screen, which is the object the user relates to while writing. In the second case, we are interested in the underlying mathematical object which is used for the subsequent analysis.

When we draw/visualise a stroke in the first, qualitative situation we will do this via a brush stroke (created in the graphics software *Krita*). In the second, quantitative one we will talk about stroke as an ordered list of points. So we will show them as small circles representing these point connected by line segments indicating both the order of the points and the picture on the screen when the stroke is properly rendered.[22] See Figure 1.7.

When we draw strokes, and we deform or change them in any way, the original stroke will be blue while the alterations will be orange.

[22]In general, we will not indicate which point is the first or the last. So the writing order is ambiguous; but it will be minuscule in the context of these images.

2 A mathematical model of handwriting

... if you want to talk about a person walking, you have to also describe the floor, because people don't just dangle their legs around in empty space.
— Alan Watts, *Out of your Mind*

This chapter deals with the general set-up of this thesis. As alluded to in Section 1.3, handwriting recognition is, in its essence, a purely practical problem. In the following chapters, however, we will explore mathematical properties of various aspects of strokes and, therefore, we will assume specific situations. So, the ideas and results here might not be transferable one-to-one to an actual HWR program.

2.1 Mathematical fundamentals

To start, we will give a self-contained introduction to Projective Geometry, Fuzzy Logic and Formal Concept Analysis. Almost all of this information can be found in [55], [33] and [15], respectively, but we will include it here explicitly since we will use it to various degrees in the subsequent chapters. Readers who are familiar with the notion of

— homogeneous coordinates (used for the general geometric set-up),

— fuzzy sets (used as a valuation function/likelihood predictor for various statements) and

— contexts, concepts and attribute exploration (used to find decision rules for the classifier),

should feel free to skip the next few pages and start directly with Section 2.2 about the basic model of handwriting recognition.

2.1.1 Projective Geometry

There are many different entry points into Projective Geometry. For example, from a purely algebraic point of view, projective spaces are sets of one-dimensional subspaces in a linear vector space over any field. Here, however, we want a more geometric and descriptive approach.

First of all, for handwriting recognition we are only interested in the real plane $\mathbb{R}^2$ — here called the **drawing plane** — since this models the touchscreen we write on. Then we ask the practical question: How can we determine the intersection of two lines without having to consider the separate case of the lines being parallel? The answer to this is simple: We add "points at infinity" which are then precisely the intersection points of parallel lines. Again, we could make this construction on a conceptual level by adding an abstract point for every bundle of parallel lines to the plane and then explain how the geometry of this new plane-like object looks like. This is done, for example, in Foundations Of Geometry. Here, we choose a more concrete way instead.

Imagine the plane $\mathbb{R}^2$ embedded into $\mathbb{R}^3$ as any affine plane which does not contain the origin. In particular, we can look at the embedding

$$(x,y) \mapsto \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

I.e., the plane we embed to is $z = 1$, and we call this the **standard embedding**, which we will use from now on. The act of converting 2D-coordinates to 3D-vectors is called **homogenisation**.

The crucial part now is that we identify all scalar multiples $\lambda P,\ \lambda \in \mathbb{R}^\times$ of this vector P with the same point. I.e. we represent any point in the embedded plane with the line running through it and the origin. We can retrieve the original Euclidean point simply via **de-homogenisation**

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightsquigarrow \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{pmatrix} \rightsquigarrow \left(\frac{x}{z}, \frac{y}{z}\right).$$

This is illustrated in Figure 2.1.

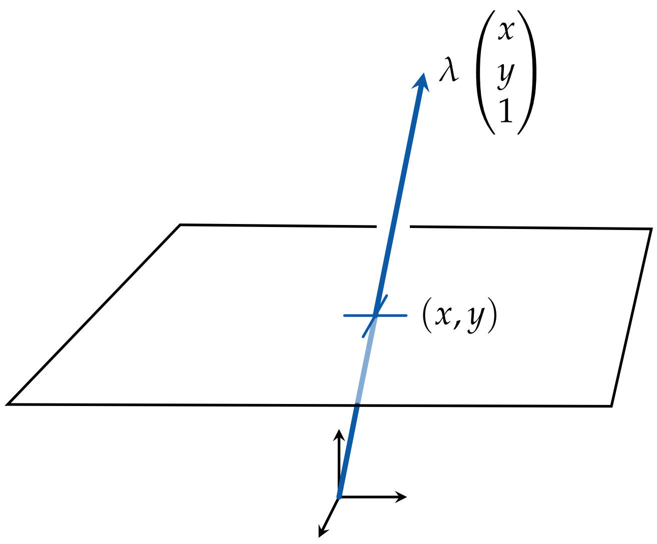


Figure 2.1: *An embedding of the projective plane.*

In particular, this means that every line through the origin in $\mathbb{R}^3$ that intersects the embedded plane represents a point in our original drawing plane. This leaves the question: What do lines parallel to this plane represent? These are precisely the **points at infinity** we talked about before. Lines in $\mathbb{R}^3$ given by $\left(\begin{smallmatrix} x \\ y \\ 0 \end{smallmatrix}\right)$ (or any non-zero multiple of it) can be imagined to intersect the plane $z = 1$ in a point which lies in direction (x, y), but infinitely far away. This can be demonstrated by considering the vector $\begin{pmatrix} tx \\ ty \\ 1 \end{pmatrix}$ for a $t \in \mathbb{R}\backslash\{0\}$. By definition it represents the same point as $\begin{pmatrix} x \\ y \\ \frac{1}{t} \end{pmatrix}$. On the one hand, when t approaches infinity this vector converges towards $\left(\begin{smallmatrix} x \\ y \\ 0 \end{smallmatrix}\right)$ and, on the other hand, the associated point (tx, ty) in the drawing plane moves towards infinity in direction (x, y). Combining the above ideas, we can consider the set

$$\mathcal{P}_{\mathbb{R}} = \frac{\mathbb{R}^3\backslash\{0\}}{\mathbb{R}^\times}$$

of all vectors in $\mathbb{R}^3$ modulo scalar multiples. The elements of $\mathcal{P}_{\mathbb{R}}$ are equivalence classes

$$[P] = \left\{\lambda P \;\middle|\; \lambda \in \mathbb{R}^\times\right\},$$

however, we will usually omit the square brackets and only write the representatives. If the last coordinate is zero, it represents a point at infinity, and if it is non-zero, it represents a finite point from our original drawing plane.

Note, that most constructions below utilise explicit representatives and, therefore, we have to check whether they are unaffected by re-scaling of these representatives.

Next, we do something similar for lines: Every line in $\mathbb{R}^2$ is given by an equation of the form

$$ax + by + c = 0.$$

We associate the non-zero vector $\left(\begin{smallmatrix} a \\ b \\ c \end{smallmatrix}\right)$ with it.[1] Scaling the equation by a non-zero real number does not change its solution, so we can, again, represent the

[1] It is better and more sound to use elements of $\mathrm{Hom}(\mathbb{R}^3, \mathbb{R})$ or at least row vectors here. But the description we give here will suffice.

line by any scalar multiple of the vector $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$.

In $\mathbb{R}^3$, we can consider the plane spanned by the origin and such a line within the embedding plane $z = 1$. Then, $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ is just one of its normal vectors. And, the other way around, any such vector $\neq 0$ represents its normal plane which intersects the embedding plane giving a line in the original drawing plane. The only vector which does not intersect the plane is $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$, and we call the line it represents the **line at infinity**. With a similar argument as above for the points, $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ is the normal vector of the x, y-coordinate plane which is parallel to $z = 1$ and hence they do intersect "infinitely far away".

This leads to the set

$$\mathcal{L}_{\mathbb{R}} = \frac{\mathbb{R}^3 \backslash \{0\}}{\mathbb{R}^\times}$$

representing all lines, in which all represent finite lines form our drawing plane except for $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$.

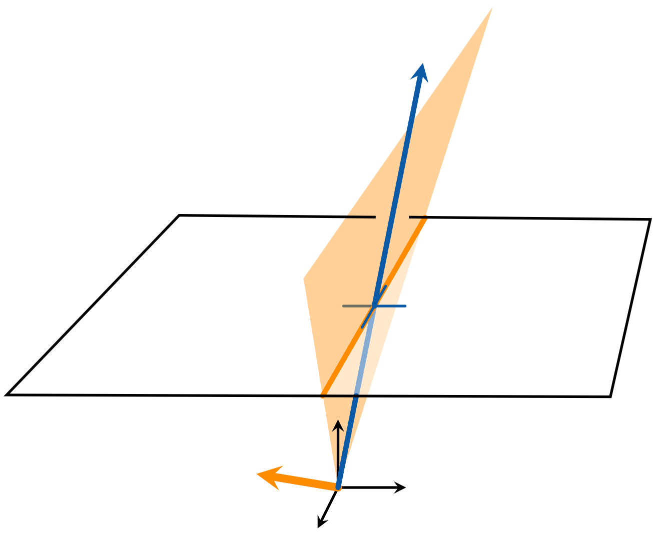


Figure 2.2: *Incidence in the projective plane.*

The beauty of these definitions starts to show when we consider the incidence relation of points and lines. A finite point given by $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ lies on a finite line $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ if and only if $ax + by + c = 0$. Moreover, this is compatible with the vectors in $\mathbb{R}^3$ representing our geometric objects: $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ represents a line through the origin and $\left(\begin{smallmatrix} a \\ b \\ c \end{smallmatrix}\right)$ represents the plane perpendicular to it. This line lies in this plane if and only if $\left(\begin{smallmatrix} x \\ y \\ 1 \end{smallmatrix}\right)$ is perpendicular to $\left(\begin{smallmatrix} a \\ b \\ c \end{smallmatrix}\right)$. Thus the same holds for their intersections with the embedding plane, which are the points and lines in the drawing plane we are actually interested in. You can see this in Figure 2.2.

It is a natural choice to extend this to all points and lines: For any $P \in \mathcal{P}_{\mathbb{R}}$ and $l \in \mathcal{L}_{\mathbb{R}}$ we say that P **lies on** l if

$$P^T l = 0.$$

Note that this definition is independent of the choice of the representatives. I.e., we can scale P and l by any non-zero number without changing the fact whether their scalar product is zero or not. In particular we get the proposition that every point at infinity lies on the line at infinity. The reason is simply that

$$\begin{pmatrix} x & y & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = 0.$$

All the above considerations give rise to the following definition.

Definition 2.1.1: The **real projective plane** is the triple

$$\mathbb{R}\mathrm{P}^2 := (\mathcal{P}_{\mathbb{R}}, \mathcal{L}_{\mathbb{R}}, \mathcal{I}_{\mathbb{R}})$$

with $\mathcal{I}_{\mathbb{R}} \subset \mathcal{P}_{\mathbb{R}} \times \mathcal{L}_{\mathbb{R}}$ a relation given by $P\mathcal{I}_{\mathbb{R}}l \iff P^T l = 0$. This we call the **incidence relation** of $\mathbb{R}\mathrm{P}^2$. △

A key feature of projective planes is that every two points have a unique line connecting them and every two lines have a unique intersection point. When projective planes are constructed as above, these connections and intersections can be explicitly computed.

Lemma 2.1.2:

— *Let $P, Q \in \mathcal{P}_{\mathbb{R}}$ be distinct points. Then $P \vee Q := P \times Q \in \mathcal{L}_{\mathbb{R}}$ is the unique line on which both these points lie.*

— *Let $l, m \in \mathcal{L}_{\mathbb{R}}$ be distinct lines. Then $l \wedge m := l \times m \in \mathcal{P}_{\mathbb{R}}$ is the unique point which lies on both these lines.*

The next major point in understanding this projective construction is to define structure-preserving maps on it — with the incidence relation being the structure that should be preserved. We want to think of them as perspective distortions — e.g., what happens when one looks at a chess board from an angle. This directly leads to the following definition.

Definition 2.1.3: A **projective transformation** is a map given by a matrix $M \in \mathrm{GL}_3(\mathbb{R})$ which operates on $\mathcal{P}_{\mathbb{R}}$ via $P \mapsto MP$ and on $\mathcal{L}_{\mathbb{R}}$ via $l \mapsto \left(M^{-1}\right)^T l$. △

Note that matrices that are scalar multiples of each other induce the same map on $\mathcal{P}_{\mathbb{R}}$ and $\mathcal{L}_{\mathbb{R}}$ and, consequently, the same projective transformation.

That this is indeed structure-preserving is easy to see: We have

$$(MP)^T \left(\left(M^{-1}\right)^T l\right) = P^T \underbrace{M^T \left(M^{-1}\right)^T}_{=I_3} l = P^T l$$

for all points P, lines l and projective transformations M. So, any point is incident to a line if and only if the same holds for the images under M.

Proposition 2.1.4: *Let A, B, C, D and A', B', C', D' be two point quadruples, each in general position. That means that no three points of a quadruple lie on a common line. Then there exists a unique projective transformation M such that*

$$MA = A', \quad MB = B', \quad MC = C', \quad MD = D'.$$

The last proposition is not surprising. A projective transformation has nine coordinates/entries and hence eight degrees of freedom since we can scale the matrix by scalars. Similarly, every point has two degrees of freedom and so the above four equations impose two linear conditions each on the entries of a projective transformation.

This is all we can and will say about the basic structure of the real projective plane for now. Before we continue with the next topic, however, we will shortly talk about two notions which we will use in this thesis.

First, there are higher-dimensional projective spaces $\mathbb{RP}^{\mathrm{d}}$, for an integer $d \geq 1$. Their point sets are defined analogously as

$$\frac{\mathbb{R}^{d+1}\backslash\{0\}}{\mathbb{R}^{\times}}.$$

The dual objects there are then given by hyperplanes and not by lines. But we will only use the point and the basic fact that scalar multiples of vectors represent the same point again.

Second, for vectors $X_1, ..., X_n \in \mathbb{R}^n$ we set

$$[X_1, ..., X_n] := \det \begin{pmatrix} | & | & \cdots & | \\ X_1 & X_2 & \cdots & X_n \\ | & | & \cdots & | \end{pmatrix},$$

with these n vectors as the columns of the matrix. Mostly, we will be interested in the case $n = 3$, i.e., when we deal with representatives of points in $\mathbb{RP}^2$.

A variant of the first fundamental theorem of projective invariant theory states that every projectively invariant function of point configurations in $\mathbb{RP}^2$ can be expressed via such determinants with the points appearing as the columns of the determinants.[2] The points we deal with — as discretisations of strokes — will never be in a stable relation to each other. Nevertheless, we will use this theorem as one motivation in Chapter 5 when we describe one possible characterisation of the shape of strokes.

The central concept in understanding why determinants of points are predominant in projective geometry is the following lemma.

[2]See [55] for more on that.

Lemma 2.1.5: *Let A, B, C be three finite points in $\mathbb{RP}^2$ in standard embedding. I.e., their last coordinate is equal to 1. Then, $\frac{1}{2}[A, B, C]$ is the signed area of the triangle ABC.*

Idea of proof. We can use a Euclidean transformation to map A to the origin and B onto the x-axis and then the statement can be shown by computing the determinant explicitly. □

A consequence of this lemma is that three points A, B, C are collinear if and only if $[A, B, C] = 0$.

In a general projective setting, it makes little sense to use this interpretation of determinants as some points in the determinant might lie at infinity. However, it is possible to generalise statements and proofs in the Euclidean plane to the projective one via determinants.

As we will use Projective Geometry mainly to describe our data, the basic definitions presented are everything we need.

2.1.2 Fuzzy Logic

Fuzzy Logic concerns itself to describe properties that do not hold with certainty — the prime example here in this section will be whether a person can be considered old and how that can be modelled based on age.

And of course, this notion of uncertainty is also a prominent phenomenon in handwriting recognition (and other applications of pattern matching and machine learning): We will rarely be 100% sure that a line drawn on a touch surface is, say, the number `3`, but we will be able to give a gauge or likelihood for how much the recorded stroke resembles a `3`.

All definitions and propositions here in this section can be found in [33].

We replace the concept of whether "a property holds" by "being the element of a set" because it is much more palpable. Moreover, we will also describe sets by functions, which makes it much easier to introduce a level of ambiguity.

Given any **base set** or **universal set** X there is the well-known bijection

$$\begin{aligned} \mathcal{P}(X) &\longrightarrow \{0,1\}^X \\ A &\longmapsto \begin{pmatrix} X & \to & \{0,1\} \\ x & \mapsto & \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases} \end{pmatrix}. \end{aligned}$$

It maps any subset to a function — which is sometimes called the **characteristic function** or **indicator function** of A — which simply encodes whether an element of X lies in A by the integers 0 and 1. When we now want to generalise the notion of a set to account for uncertainty we can do that easily with indicator functions.

Definition 2.1.6: Given a base set X, a **fuzzy (sub-)set** of X is a function $m : X \to [0,1]$. △

Every element $x \in X$ with $m(x) = 1$ is thought to be definitely an element of this "set" and an element $m(x) = 0$ is thought to be definitely not in the set. For all other element the values in-between 0 and 1 describe varying degrees of "maybe". To make it a bit easier to talk about this new definition of sets one usually gives a name to the abstract set, say A, and calls the associated function $m_A : X \to [0,1]$ the **membership function** of A. With this we can make statements like

Element x mostly likely lies in A, because $m_A(x) = 0.99999$.

Before we illustrate fuzzy sets by an example, a few more basic definitions:

Definition 2.1.7: Let X be a base set and A a fuzzy subset of it. The **complement** $\bar{A}$ of A is defined by the membership function $m_{\bar{A}}(x) := 1 - m_A(x)$. The fuzzy set A is called a **crisp** set if $m_A(X) \subseteq \{0,1\}$. △

So, crisp sets are just ordinary sets for which the membership function takes the shape of an indicator function. Moreover, the definition of the complement to any fuzzy set is very natural in the sense that it is compatible with indicator function of ordinary sets. We will see similar constructions soon.

Example 2.1.8: Let $X = \mathbb{R}_0^+$ be the base set describing the age of a human in years. Then we can consider the fuzzy set OLD with the following membership function:

$$m_{\text{OLD}}(x) := \begin{cases} 0, & \text{if } \; x \leq 20, \\ \frac{1}{40}x - \frac{1}{2}, & \text{if } \; 20 < x \leq 60, \\ 1, & \text{if } \; 60 < x. \end{cases}$$

Its graph can be seen in Figure 2.3.

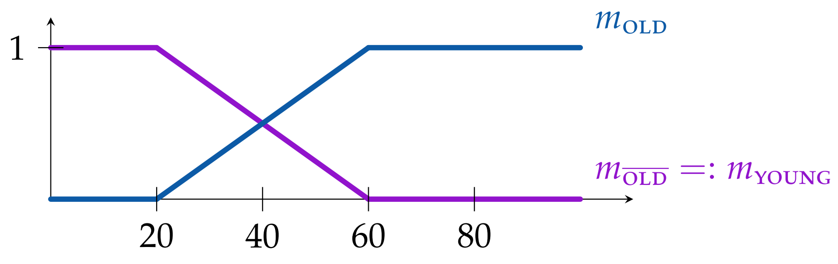


Figure 2.3: *A simple function modelling how* OLD *or* YOUNG *someone is.*

This is a very crude and simplistic model to describe when a person is OLD. With a life expectancy of roughly 80 years it is at least somewhat reasonable to declare the upper quartile as OLD and the lower quartile as $\overline{\text{OLD}} =:$ YOUNG. So it makes sense that $m_{\text{OLD}}(40) = 0.5$ with the graph of the membership function being centrosymmetric with respect to this point. We could say that a person who is 40 years is both OLD and YOUNG at the same time; or neither of them. So, this model might be very simplistic, but at least it is not entirely flawed. However, we can improve it.

Note that the above function is piecewise linear which implies that 20 years and 60 years are hard transition points for being OLD and YOUNG. To allow for more flexibility we consider the following (smooth) S-shaped function:

$$m_{\text{OLD}}(x) := \begin{cases} \frac{1}{3200}x^2, & \text{if } \; x \leq 40, \\ -\frac{1}{3200}(x-80)^2 + 1, & \text{if } \; 40 < x \leq 80, \\ 1, & \text{if } \; 80 < x. \end{cases}$$

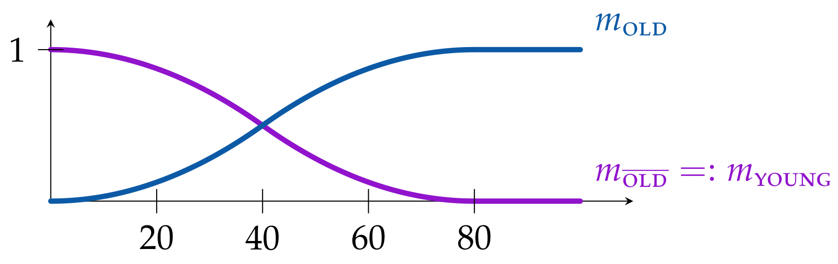


Figure 2.4: *Another function modelling how* OLD *or* YOUNG *someone is.*

Again a symmetric function, which is shown in Figure 2.4. This is already better, since the OLD-ness of a person gradually increases with age.

Lastly, one could also say that getting OLD does not even start before turning 60, i.e.,

$$m_{\text{OLD}}(x) := \begin{cases} 0, & \text{if } \ x \leq 60, \\ \frac{1}{200}(x-60)^2, & \text{if } \ 60 < x \leq 70, \\ -\frac{1}{200}(x-80)^2+1, & \text{if } \ 70 < x \leq 80, \\ 1, & \text{if } \ 80 < x. \end{cases}$$

Applying the same logic to being YOUNG results in

$$m_{\text{YOUNG}}(x) := \begin{cases} -\frac{1}{200}x^2+1, & \text{if } \ x \leq 10, \\ \frac{1}{200}(x-20)^2, & \text{if } \ 10 < x \leq 20, \\ 0, & \text{if } \ 20 < x. \end{cases}$$

These functions are illustrated in Figure 2.5. We see that $\overline{\text{OLD}}$ is not the same as YOUNG; in contrast to what we assumed in the first two versions above. This makes sense even in colloquial English, since YOUNG is not the complementary attribute to OLD, but its polar opposite. This shows that one must be careful modelling real-world phenomena via fuzzy sets. △

The attribute names OLD and YOUNG used in the example above to evaluate, judge or organise the elements of the base set via fuzzy values are called **linguistic variables**. They allow the classification of elements via a natural language which makes decision rules derived from them in practice much more

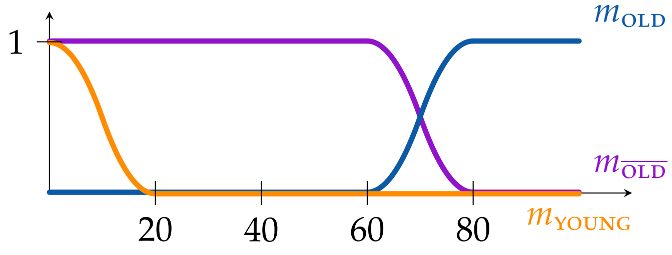


Figure 2.5: *Asymmetric membership functions.*

comprehensible. E.g., saying

Person x is rather old

is usually much more intelligible that

Person x's age has a membership value of 0.9 *with respect to the fuzzy set* OLD.

And here we see the applicability for handwriting recognition, as we want to make precisely such qualitative statements like

this recorded stroke looks more like a 3 than a 7,

which we now can easily model via fuzzy sets.

There is a lot more to say about Fuzzy Logic. For our purposed though, the basic definition above is almost enough. What is still missing is how we can combine fuzzy sets. More precisely, is there a way to intersect and join fuzzy sets, such that for crisp sets we get the usual operations? To answer that, we introduce the concept of t-norms.

Definition 2.1.9: A **t-norm** is a binary operator $\odot : [0,1] \times [0,1] \to [0,1]$ with the following properties:

— **Commutativity**: $x \odot y = y \odot x$ for all $x, y \in [0,1]$.

— **Associativity**: $(x \odot y) \odot z = x \odot (y \odot z)$ for all $x, y, z \in [0,1]$.

— **Monotonicity**: $a \odot b \leq c \odot b$ for all $a, b, c \in [0,1]$ with $a \leq b$.

— The number 1 is an **identity element**: $1 \odot x = x$ for all $x \in [0,1]$.

$\triangle$

Note that monotonicity also holds in the second argument due to the commutativity. The prime example for a t-norm is the ordinary product $x \cdot y$ of two real numbers — and that is the reason for the choice of the symbol $\odot$. Other important examples are

— the minimum t-norm $(x, y) \mapsto \min\{x, y\}$,

— the Łukasiewicz t-norm $(x, y) \mapsto \max\{0, x + y - 1\}$ and

— the Hamacher t-norms $(x, y) \mapsto \begin{cases} 0, & \text{if } p = x = y = 0, \\ \frac{xy}{p+(1-p)\cdot(x+y-xy)}, & \text{else,} \end{cases}$
for any $p \in \mathbb{R}_0^+$.

The last one is itself a direct generalisation of the ordinary product (choose $p = 1$), it is the only t-norm given by a rational function and it is related relativistic physics.

Fuzzy values share many properties with probabilities even though they are technically vastly different things. However, after all, the interpretations of statements like

The element x is in a fuzzy set with an 80% *likelihood*

and

The event A occurs with an 80% *probability*

are very similar. So it is no surprise that many ideas in Fuzzy Logic are modelled not only after classical Set Theory but also after Probability Theory. In particular, the probability of two (independent) events happening at the same time is equal to the product of the individual probabilities. This leads to the idea that the likelihood of an element being in two fuzzy sets should be the product of the respective membership functions.

In practice, this might lead to unsatisfying results depending on what the fuzzy sets model. And this is where t-norms come into the picture: as a generalisation of the product of two real numbers.

Definition 2.1.10: Given two fuzzy sets A, B on the same base set X and a t-norm $\odot$, the **intersection** $A \cap B$ of A and B (with respect to $\odot$) is given by the membership function

$$m_{A\cap B}(x) := m_A(x) \odot m_B(x).$$

$\triangle$

Before we show some examples, we give two fundamental properties of t-norms.

Proposition 2.1.11: *Let $\odot$ be any t-norm. Then the following properties hold:*

(1.) For all $x, y \in [0,1]$ we have

$$x \odot y \leq \min\{x, y\}.$$

(2.) For all $x \in [0,1]$ we have

$$0 \odot x = 0.$$

Note how the second property above makes sense in our context of generalised sets: If an element x is not in a (fuzzy) set A, it will not be in an intersection of A with another (fuzzy) set.

There are many more properties of t-norm — e.g. whether they are continuous or have the Archimedean property[3] — but these two fundamental ones are enough for our considerations.

Example 2.1.12: Computing the t-norm of two given membership functions of fuzzy sets is, in general, tedious and not very insightful. So, in order to illustrate the intersection of fuzzy sets, let us look at their graphs and describe the intersection membership function only qualitatively.

Consider the base set $X = \mathbb{R}$ and a fuzzy set A given by a triangular membership function and a fuzzy set B given by a bell-shaped membership function in

[3]This means that for all $x, y \in (0,1)$ there exists an $n \in \mathbb{N}$ such that $\underbrace{x \odot \ldots \odot x}_{n \text{ times}} \leq y$.

Figure 2.6.

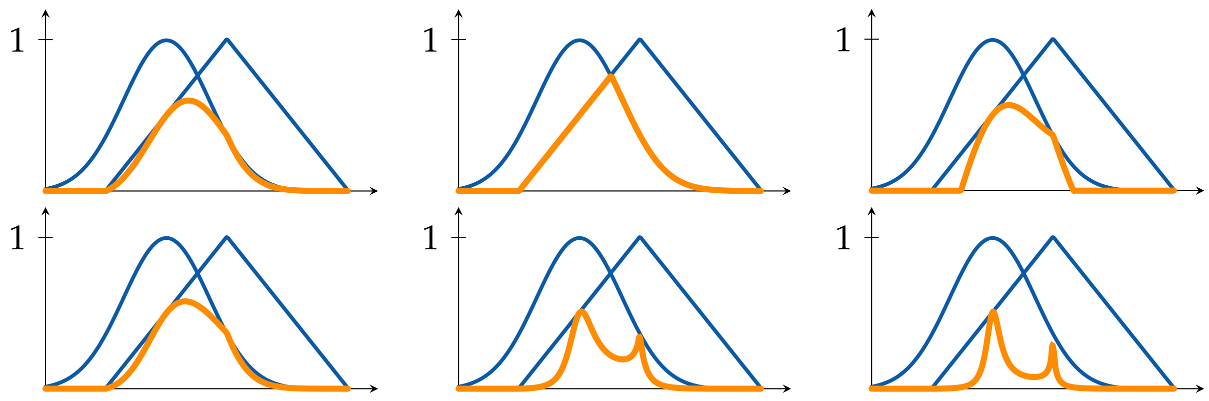


Figure 2.6: *Graphs of fuzzy sets and their intersection w.r.t. six different t-norms. Top row: product (left), minimum (middle) and Łukasiewicz norm (right). Bottom row: Hamacher norm for $p = 2$ (left), $p = 30$ (middle) and $p = 100$ (right).*

Observe the intersection with respect to the minimum norm. It is given by the intersection of the areas limited by the x-axis and the graphs of the membership functions. This is the reason why the intersection of fuzzy sets is often defined by the minimum as the canonical way. Moreover, this allows for an interpretation of the area below the graphs as the actual fuzzy set. △

In order to define the union of fuzzy sets, we can do two things: Define a dual concept to t-norm, called **t-conorm**, or use De Morgan's laws from Set Theory and consider the map

$$(x, y) \quad \mapsto \quad 1 - ((1 - x) \odot (1 - y)).$$

As it turns out, the results are the same. For the range of this thesis, however, we do not need that. We will use fuzzy sets to describe various uncertain geometric properties and determine the likelihood of combinations of them via t-norms.

2.1.3 Formal Concept Analysis

Formal Concept Analysis (FCA for short) is an all-purpose data analysis and data visualisation technique that can be very useful, given the right circumstances. The basic idea is to describe objects by specific attributes and then analyse what a combination of certain attributes can say about the corresponding objects.

This is precisely what we want to do in handwriting recognition: We have lines drawn on a touchscreen, we measure geometric properties of them and then we want to make deductions on how the lines look by just considering the measured features.

All definitions and propositions here in this section can be found in [15].

Definition 2.1.13: A **(formal) context** is a triple (G, M, I) of a set G of **objects**, a set M of **attributes** and an **incidence relation** $I \subset G \times M$. $\triangle$

Subsequently, we will call these objects just 'contexts' and not 'formal contexts'. This is also true for all other definitions here in this section: There are formal objects, formal attributes, formal concepts, formal implications, and so on. The adjective 'formal' is added, because all these terms carry much meaning in colloquial speech, especially in the areas of data analysis FCA is mostly used in. However, for the purely mathematical considerations here, we drop it.

The incidence relation of a context is easily depicted as a table in which a cell has an 'x' if and only if the object of its row and the attribute of its column are together in the relation.

Example 2.1.14: Consider the mathematicians

$$G = \{\texttt{Gauss},\ \texttt{Noether},\ \texttt{Faltings},\ \texttt{Euclid},\ \texttt{Jordan}\}$$

as objects and the attributes

$$\begin{aligned} M = \{&\textsc{was born after } 1000 \text{ CE}, \quad \textsc{is male}, \quad \textsc{is German}, \\ &\textsc{has a Fields medal}, \quad \textsc{has a theorem named after them}\}. \end{aligned}$$

Then the table (with shortened attribute names)

	1000 CE	MALE	GERMAN	FIELDS	THEOREM
Gauss	x	x	x		x
Noether	x		x		x
Faltings	x	x	x	x	x
Euclid		x			x
Jordan	x	x			x

represents a context (G, M, I). △

After defining a structure in mathematics, the obvious follow-up question is, how the sub-structures look like. For any context we can easily define sub-contexts by subsets of the object set or the attribute set. But this just means looking at sub-tables and it is neither handy nor interesting. We get the proper sub-structures by formalising how sets of objects and attributes relate to each other.

Definition 2.1.15: Let (G, M, I) be a context and define the two maps

$$' : \mathcal{P}(G) \to \mathcal{P}(M), \quad A \mapsto \{m \in M \mid \forall g \in A : gIm\}$$

and

$$' : \mathcal{P}(M) \to \mathcal{P}(G), \quad B \mapsto \{g \in G \mid \forall m \in B : gIm\}$$

They are called **derivations**. △

They describe the attributes a set of objects have in common and, dually, the objects in which a set of attributes appears together. The most important property of these maps for us is the following.

Proposition 2.1.16: *The maps $'' : \mathcal{P}(G) \to \mathcal{P}(G)$ and $'' : \mathcal{P}(M) \to \mathcal{P}(M)$, obtained from concatenating the derivations above, are hull (or closure) operators. I.e., they fulfil the following properties for all $X, Y \subseteq G$ or $X, Y \subseteq M$.*

— ***Extensiveness:*** *$X \subseteq X''$.*

— ***Idempotence:*** *$X'''' = X''$.*

— ***Monotonicity***: $X \subseteq Y \implies X'' \subseteq Y''$.

Now, the back-and-forth between objects and attributes via the derivations gives us the desired sub-structures.

Definition 2.1.17: Let (G, M, I) be a context and $A \subseteq G$ and $B \subseteq M$. Then the pair (A, B) is called a **concept** of the context if $A' = B$ and $B' = A$. In this case, A is called the **extent** and B the **intent** of this concept. The set of all concepts of the context (G, M, I) is denoted by $\mathfrak{B}(G, M, I)$. For single objects $g \in G$ and attributes $m \in M$ we write g' and m' instead of $\{g\}'$ and $\{m\}'$, respectively, and we call g' an **object intent** and m' an **attribute extent**. △

This means, (A, B) is a concept if the objects in A have exactly the attributes in B in common and the attributes in B appear together exactly in the objects in A. I.e., in a concept, the objects uniquely describe the attributes and vice-versa.

There are many properties of concepts. The next proposition lists the most important and fundamental ones.

Proposition 2.1.18: *Let $\mathcal{C} = (G, M, I)$ be a context.*

— *The derivations in $\mathcal{C}$ reverse the order of sets. I.e.,*

$$A_1 \subseteq A_2 \implies A_1' \supseteq A_2'$$

for any subsets A_i of either G or M.

— *Let (A, B) be a concept of $\mathcal{C}$. Then, A and B are hulls. I.e., $A = A''$ and $B = B''$.*

— *When we order concepts of $\mathcal{C}$ via*

$$(A_1, B_1) \leq (A_2, B_2) \quad :\Longleftrightarrow \quad A_1 \subseteq A_2,$$

the set of concepts with this order $(\mathfrak{B}(G, M, I), \leq)$ is a complete lattice. That means it is a partially ordered set, in which supremum and infimum of arbitrary *sets exist. Explicitly, they are*

$$\bigvee_{t\in T}(A_t,B_t)=\left(\left(\bigcup_{t\in T}A_t\right)'',\bigcap_{t\in T}B_t\right)$$

and

$$\bigwedge_{t\in T}(A_t,B_t)=\left(\bigcap_{t\in T}A_t,\left(\bigcup_{t\in T}B_t\right)''\right),$$

respectively.

Example 2.1.19: Looking back at our context of mathematicians from Example 2.1.14

	1000 CE	MALE	GERMAN	FIELDS	THEOREM
`Gauss`	x	x	x		x
`Noether`	x		x		x
`Faltings`	x	x	x	x	x
`Euclid`		x			x
`Jordan`	x	x			x

we can consider the objects $A=\{$`Gauss, Noether`$\}$. Then,

$$A'=\{\text{1000 CE, GERMAN, THEOREM}\},$$

as these are the attributes Gauss and Noether have in common.

Moreover,

$$A''=\{\texttt{Gauss, Noether, Faltings}\}.$$

This means, that not only Gauss and Noether are German mathematicians, born in the last millennium with a theorem named after them, but Faltings is one, too. These three properties describe exactly these three mathematicians; at least here in this context. So,

$$(\{\texttt{Gauss, Noether, Faltings}\},\ \{\text{1000 CE, GERMAN, THEOREM}\})$$

is a concept of this context. △

When applying FCA to actual data one rarely has attributes that fall under a dichotomy of 'present' and 'not present'. Often there are real-valued quantities to consider like price, weight or volume or attributes with several different types like colours or names of suppliers. In order to get ordinary contexts as we introduced them above, one has to make a few additional definitions.

Definition 2.1.20: A **multi-valued** context is a quadruple (G, M, W, I) with objects G, attributes M, a **value set** W and an incidence relation $I \subset G \times M \times W$, such that for all $g \in G$ and $m \in M$ the implication

$$(g, m, w_1) \in I \ \wedge \ (g, m, w_2) \in I \quad \Longrightarrow \quad w_1 = w_2$$

holds. △

This condition seems a bit technical at first. It is a lot easier to view the attributes in a multi-valued context as functions $G \to W$ and then this condition just means that these functions are well-defined. And with this functional view in mind we often write $m(g) = w$ instead of $(g, m, w) \in I$. Unsurprisingly, multi-valued contexts are once again displayed as tables.

Example 2.1.21: Considering the objects/books

$$G = \{\texttt{Lord Of The Rings}, \ \texttt{Atlas Shrugged}\},$$

the attributes

$$M = \{\text{PRICE}, \quad \text{PAGES}, \quad \text{AUTHOR}\},$$

and the values

$$W = \{\text{Tolkien}, \quad \text{Rand}\} \quad \cup \quad \mathbb{R},$$

the following table represents a multi-valued context (G, M, W, I):

	PRICE	PAGES	AUTHOR
`Lord Of The Rings`	19.98	1134	Tolkien
`Atlas Shrugged`	7.89	1069	Rand

△

Such a multi-valued context can and will be transformed into an ordinary one via a process called **scaling**. It is one of the most important steps in modelling a useful context from actual data.

Definition 2.1.22: Given a multi-valued context (G, M, W, I), a **scale** for an attribute $m \in M$ is a context $S_m = (G_m, M_m, I_m)$ with $m(G) \subseteq G_m$. $\triangle$

What this means is, that the values (or ranges of values) of W get turned into new attributes. However, instead of explaining this process in general, we will consider two illustrative examples.

Example 2.1.23: Consider the following multi-valued context that models cars and some of their properties.

	PRICE	NUMBER OF SEATS	MANUFACTURER
`Skyline`	20500	4	Nissan
`Ford GT`	300000	2	Ford
`Fiesta`	13000	4	Ford

Now we consider every value that can occur for the price, the number of seats and the manufacturer as a new attribute. For the (more or less continuous) values which describe the price, we use adequate bins/intervals for the attributes. So, instead of saying that the PRICE value of a `Skyline` is 20500 we say that the object `Skyline` has the attribute PRICE BETWEEN 10000 AND 100000.

After reformulating the NUMBER OF SEATS and the MANUFACTURER in a similar way, we might get:

	PRICE / 1,000			SEATS					MANUFACTURER		
	0–10	10–100	>100	1	2	3	4	5	NISSAN	BMW	FORD
`Skyline`		x					x		x		
`Ford GT`			x		x						x
`Fiesta`		x					x				x

Note that we could drop the third manufacturer BMW since no car has it as an attribute. We did include it in the first place to illustrate that scaling might introduce values that were not present before.

In subsequent analyses we might ask questions like

Can three people sit in the car?

But in the last context, there is no connection between the different attributes encoding the number of seats. In reality, three people can sit in any car that has three or more seats. That means that there is an internal order for the number of seats: If a car has four seats, it also has three seats. In order to account for this, we use an **ordinal scale** to describe this attribute. That means that

— if we have a value range in the multi-valued context that is linearly ordered and

— if an object in the scaled context has an attribute m,

then it also shall have all attributes that are smaller than m; or larger, depending on what they model. We can do this for the number of seats and, of course, for the price, too. Against prevalent belief, car manufacturers cannot be ordered, so for these attributes, we do not use an ordinal scale.

	PRICE / 1,000			SEATS					MANUFACTURER		
	0–10	10–100	>100	1	2	3	4	5	NISSAN	BMW	FORD
`Skyline`	x	x		x	x	x	x		x		
`Ford GT`	x	x	x	x	x						x
`Fiesta`	x	x		x	x	x	x				x

A different scale we can use is the **dichotomic scale**. It encodes yes-no relations and can be applied to every context. For multi-valued context like the one above this means to compress each range of values into two bins. E.g. whether the price is below or above 50,000 or whether the manufacturer is Nissan or not.

However, we can apply a dichotomic scale even to regular (single-valued) contexts. Consider the simple (and unspecified) example below.

	m_1	m_2	m_3
g_1	x		x
g_2	x	x	x
g_3		x	

For every attribute m_i we introduce the complementary attribute $\overline{m_i}$ which shall be present in an object if and only if m_i is not. This results in:

	m_1	$\overline{m_1}$	m_2	$\overline{m_2}$	m_3	$\overline{m_3}$
g_1	x			x	x	
g_2	x		x		x	
g_3		x	x			x

There are many more ways to scale a (multi-valued) context, see [15], but these two are the ones we will use in this thesis. △

With these constructions, we can model internal relations between attributes that might have gotten lost during the transition from multi-valued to ordinary contexts. The idea that certain attributes imply the presence of other attributes can be formalised and generalised. We do this now in a very abbreviated way which is sufficient for the use in this thesis. For a thorough explanation we refer to [15] and [14] again.

Definition 2.1.24: Let $\mathcal{C} = (G, M, I)$ be a context and $A, B \subset M$ be sets of attributes. The pair (A, B) is called an **implication** and is denoted by $A \to B$. We say an implication **holds** in a context if every object that has all attributes in A also has all attributes in B. I.e., if $A' \subseteq B'$. We call A the **premise** and B the **conclusion**. △

These implications obey the same rules implications introduced in logic do. E.g. $\bar{B} \to \bar{A}$ holds if $A \to B$ holds, and $A \to C$ holds if $A \to B$ and $B \to C$ do. In particular, given a set of implications that hold in a context, we can find and build other implications that hold, too.

A main result in Formal Concept Analysis is that, for a context with a finite attribute set M, we can describe and explicitly construct a minimal generating set of implications. I.e., a set of implications that hold in a context, from which we can deduce every other implication, and which cannot be smaller. This **Duquenne-Guigues basis**, named after the discoverers of this object (see [19]), can be found via an algorithm called **attribute exploration**.

If applied to a given context, it simply produces this particular set of implications. In practice it has a much bigger significance: most contexts are a small snapshot of all possible objects. So, attribute exploration might find implications that hold for these examples, but not in general. The implications are found in a specific order which allows a user/supervisor to check every implication for universality and add objects as counterexamples if necessary.

For example, we might want to analyse certain vertebrates. If we start with a context containing wolves, bears and gorillas, attribute exploration might propose the fact that all vertebrate are mammals and have hair. Then we can add animals that are vertebrates but are not mammals or do not have hair to complete the picture of vertebrates.[4]

In handwriting recognition this might look like the following: we build a context from many samples of handwritten strokes for which we compute certain geometric properties. Due to sampling anomalies, all 3's we recorded might have a loop in the middle and that all 8's have their start point at the very top. However, there are of course ways to write a 3 without a loop and some people start writing an 8 in the middle near the intersection. Adding them allows us to refine the context before an extensive analysis.[5]

There is a lot to say about how implications in contexts work and what can be done with them. But we will be content with the basic Definition 2.1.24.

In [14] Ganter and Kuznetsov introduce the notion the hypotheses for contexts. These are special implications and we will introduce them in Section 4.2 and describe their use in ALICE:HWR.

[4]By definition, all mammals have hair. So not all combinations of these attributes and their negations can be added.

[5]Later we discuss how it might be beneficial to introduce separate categories for these different writing styles.

2.2 A base model for strokes

In Section 1.3 we introduced the general HWR problem in a relatively informal way. Here we want to model this problem mathematically stringently. As already mentioned at the beginning of this chapter, the subsequent discussion might not necessarily be transferable to praxis. We start our considerations of "theoretical" handwriting recognition by giving a very broad definition of the terms **strokes** and **features**. Afterwards, we present how these ideas form the basis for ALICE:HWR.

2.2.1 Strokes

The central objects used in (feature-based) HWR are strokes. Ideally they would be curves, i.e., continuous (or even smooth) functions $[0,1] \to \mathbb{R}^2$. This approach is used in praxis, for example by Frenkel and Basri in [12].

In this thesis, strokes will be finite sequences of points; discretising the continuous finger movement the user makes on a touch screen.[6] What constitutes a point here is up to debate and is determined by what information is recorded. On the basic level, a point consists of x- and y-coordinates, but additional things like timestamps, pressure and the azimuth of a pen/stylus can be added.

Definition 2.2.1: Given an integer $d \geq 2$, a **point** is an element of $\mathbb{R}^d$. For an integer $n \geq 3$, a **stroke** is an element of $\left(\mathbb{R}^d\right)^n$. △

The number d determines how many different parameters of a point are recorded. For the scope of this thesis, we assume that it is equal to 2 — so we assume that we only record x- and y-coordinates. The number n, called the **sampling rate** of the stroke, determines by how many points a curve is sampled. In practice, it usually ranges between 20 and 100. For us, it is crucial that it is "large enough". I.e., when specific terms do not make sense (e.g. a denominator becomes zero) when $n \leq 3$, we can ignore it, as we will usually

[6]The reason we consider the discrete case is that we want to apply our results directly. Continuous models might be good in theory, but they must be adapted to finite point sets if they should be used in praxis.

not consider strokes sampled by only three points or less.[7] But for any practical purposes, imagine it to be around 32.

When writing strokes we will alternate between various different representations. First, at the base level given by the definition, a stroke s is of the form

$$s = (P_i)_{i=1}^n$$

with every point P_i being a point with coordinates (x_i, y_i). We can write this in the form

$$s_{sep} := (x_1, x_2, \cdots, x_n, y_1, y_2, \cdots, y_n)^T,$$

listing all x-coordinates of all points first and then all y-coordinates.[8]

We might use homogeneous coordinates for points and indicate this with a hat-symbol. I.e. points $\hat{P}_i = \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$, strokes $\hat{s} = (\hat{P}_i)_{i=1}^n$ and $\hat{\mathfrak{S}} = \left\{ (\hat{P}_i)_{i=1}^n \in (\mathbb{RP}^2)^n \;\middle|\; \hat{P}_i \text{ finite} \right\}$. In these cases we always assume that the points are given in standard embedding; i.e. with last coordinate equal to 1. And we will simply view the sets $\mathfrak{S}$ and $\hat{\mathfrak{S}}$ as identical.

Moreover, if not stated otherwise, when we talk about strokes s and t, we will always assume that their points are named P_i and Q_i, respectively. Also, we always assume that a point P_i in a stroke has coordinates (x_i, y_i). This is handy, as we sometimes need the whole stroke s and sometimes the explicit x- and y-coordinates of its points.

Now, we will call the set of all strokes we consider — and which we want to use for handwriting recognition — the **stroke space** and we will denote it by $\mathfrak{S}$. For the major part of this thesis, we will concern ourselves only with one particular stroke space; namely $\mathfrak{S} = (\mathbb{R}^2)^n$. This will be our **standard stroke space** and whenever we write $\mathfrak{S}$, we will mean this set by default.

Before we continue, we will briefly present two smaller stroke spaces which represent normalised strokes. The reason to consider them is that strokes located at different positions of the touch screen or, in theory, in the whole drawing plane $\mathbb{R}^2$ might represent similar things and so it is handy to bring them to

[7]ALICE:HWR actively prevents such fringe cases by ensuring that every stroke is sampled with at least four points.

[8]The index '*sep*' here stand for 'separated'.

the same place to make comparisons easier and more straightforward.

First, we might assume that the start and end point of strokes are fixed. Second, we can normalise the bounding box of the strokes. It is the smallest axes-parallel rectangle enclosing (the points of) the stroke. It is a common object in HWR (cf. [11]), and we can normalise strokes by demanding that the bounding box has edge length 1 and that its centre sits at the origin.

Definition 2.2.2:

1. For any two different $S, E \in \mathbb{R}^2$ we define the set of **nailed strokes** by

$$\mathfrak{N}_{S,E} := \left\{ (P_i)_{i=1}^n \in \left(\mathbb{R}^2\right)^n \;\middle|\; P_1 = S \;\wedge\; P_n = E \right\}$$

2. For a stroke $s \in \mathfrak{S}$ set the points

$$\min(s) := \left(\min_{i=1,\dots,n} x_i, \min_{i=1,\dots,n} y_i \right) \qquad \text{and} \qquad \max(s) := \left(\max_{i=1,\dots,n} x_i, \max_{i=1,\dots,n} y_i \right)$$

in $\mathbb{R}^2$. Then define the **bounding box** $\mathbb{B}(s)$ of s as the axes-parallel rectangle spanned by the points $\min(s)$ and $\max(s)$.

Define the set of **centred strokes** $\mathfrak{Q}$ as

$$\mathfrak{Q} := \left\{ s \in \mathfrak{S} \;\middle|\; \min(s) = \left(-\frac{1}{2}, -\frac{1}{2}\right) \text{ and } \max(s) = \left(\frac{1}{2}, \frac{1}{2}\right) \right\}. \qquad \triangle$$

In order to model the bounding box correctly, we would need to add the condition that on every edge of the square spanned by $\left(-\frac{1}{2}, -\frac{1}{2}\right)$ and $\left(\frac{1}{2}, \frac{1}{2}\right)$ lies a point of the stroke. However, this then excludes the case of perfectly vertical and horizontal lines for which the bounding box has width and height 0, respectively.

Normalising a stroke such that it lies in one of the above sets $\mathfrak{N}$ or $\mathfrak{Q}$ can be realised by applying affine transformations, respectively, to the elements in $\left(\mathbb{R}^2\right)^n$. We will talk about them more when we apply projective transformations to strokes in Section 3.1.

Apart from that, there are many other additional conditions we might demand: In praxis we often have $\varepsilon < \|P_{i+1} - P_i\|_2 < \delta$ for suitable $0 < \varepsilon < \delta$, due to the fact that we do not want consecutive points to coincide and that we use a touchscreen of finite size. Alternatively, we might only consider strokes for which the distance between consecutive points is constant. We could also model all strokes as discretised Bézier curves, which are polynomial splines commonly used in computer graphics.

And even if we are content with using all sequences of n points as the stroke space, using $\left(\mathbb{R}^2\right)^n$ means that we sample every line recorded from the touchscreen by the same number of n points. For large screens and/or a big variance in the length of the lines it might be better to consider $\bigcup_{i=n}^{N} \left(\mathbb{R}^2\right)^i$ for reasonable choices of n and N. However, because of the application in ALICE:HWR and the general focus on letter- and number-like icons, we use a fixed sample rate. In a more general stroke and symbol recognition algorithm, it is reasonable to sort the recorded stroke into coarse categories depending on their length. Re-sampling the strokes in one such category with a specified sampling rate reduces every subsequent analysis to the situation presented here in this thesis.

The stroke space $\mathfrak{S}$ in and of itself does not tell us anything about the strokes we want to identify. It is an obvious idea to consider a subset $\mathfrak{T} \subset \mathfrak{S}$ of all strokes that represent the symbols we want to recognise. And in practice, we will do that. However, we will come to this from a particular angle.

The main problem in HWR is, as was stated in Chapter 1, the many different strokes that may represent the same abstract line. These differences can come from inherently distinct ways to write a specific stroke — e.g. a 3 with or without a loop — and user-specific writing styles — e.g. whether the "bulges" of a 3 are rather round or pointy. Both of these variations and others can be seen in Figure 2.7.

In order to model this, we assume that there is a likelihood for every stroke to represent a specific stroke type. Here and in the rest of the thesis, **stroke type** will stand for the abstract thing a user wants to draw on a touch surface — whereas strokes are then the discretised data recorded.

Definition 2.2.3: Let $\mathbb{A}$ be a finite set which we call the **alphabet**. It represents

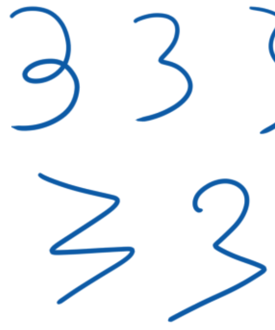


Figure 2.7: *Various ways to write* 3.

the abstract lines or stroke types drawn on a touch surface. Then, a **sample dispersal for $\mathbb{A}$ on $\mathfrak{S}$** is a set $\mathcal{D} = \mathcal{D}_{\mathbb{A},\mathfrak{S}}$ of fuzzy subsets of $\mathfrak{S}$ indexed by $\mathbb{A}$. I.e.,

$$\mathcal{D} = \{m_l : \mathfrak{S} \to [0,1] \ \mid \ l \in \mathbb{A}\}\,.$$

△

Following the general idea of fuzzy sets, we will informally call a stroke $s \in \mathfrak{S}$ a **good sample** for the stroke type $l \in \mathbb{A}$ if the value $m_l(s)$ is high and, conversely, a **bad sample** if this value is low. Take for example the two strokes in Figure 2.8.

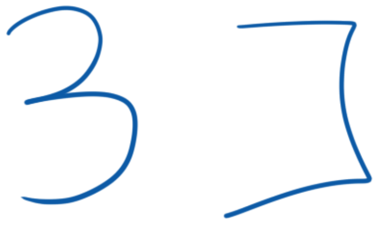


Figure 2.8: *Strokes with a different likelihoods to be a* 3.

Here we see that the left one is clearly a 3, so its fuzzy value might be $m_3(s_{\text{left}}) = 0.9$. The right one might be confused with a]-bracket by an algorithm and in any way it is at least a very sloppy 3. So, we might have $m_3(s_{\text{right}}) = 0.2$. But this depends of course on the underlying alphabet. If the only lines we want to recognise are 3 and V, then the value $m_3(s_{\text{right}})$ should probably be much higher.

Definition 2.2.4: For an alphabet $\mathbb{A}$ and a sample dispersal $\mathcal{D}$ for $\mathbb{A}$ on $\mathfrak{S}$ call a function $\mathfrak{T} : \mathbb{A} \to \mathcal{P}(\mathfrak{S})$ **a collection of good samples** if for all $l \in \mathbb{A}$ we have $m_l(\mathfrak{T}(l)) \in [\lambda_l, 1]$ for some predetermined thresholds $\lambda_l \in [0,1]$. Moreover, denote the set $\mathfrak{T}(l)$ by $\mathfrak{T}_l$. △

Since we do not know the concrete sample dispersal when we want to analyse strokes, we will only work with collections of good samples from now on. I.e., whenever we have such collection we assume that the strokes in the sets $\mathfrak{T}_l$ represent the type l well. Nevertheless, we to keep in mind that there is an underlying assessment of which strokes represent which types to which degree.

An important point for the practical application here is that two (or more) elements of the alphabet might stand for the same type. Looking at the different `3`'s in Figure 2.7 again, and depending on the particular classification method, it might be handy to see the first row as representative of one specific type `3a`, the first in the second line as a representative of a type `3b` and the very last one as `3c`. Only at the very end of the recognition process we then collapse all these cases into the type `3`.

However, it might also be the other way around: for certain analyses and classification steps, it might be handy to merge several different symbols into one. For example, we will soon see that recognising strokes which represent straight lines or closed loops is quite easy. So, when looking at the Arabic numerals `0`,...,`9` we can model them in a first step via the alphabet $\mathbb{A}_1 = \{\texttt{straight}, \texttt{loop}, \texttt{other}\}$ and then sub-classifying the loop-like symbols based on the alphabet $\mathbb{A}_{\texttt{loop}} = \{\texttt{0}, \texttt{8}\}$.

2.2.2 Features

Now, the main problem in handwriting recognition is, of course, that we do not know which strokes are good representatives for any type we are interested in. If we had the actual membership functions at our disposal for all types, we could check which line has the highest likelihood to fit the recorded stroke.

The best we can do is to approximate these likelihoods by considering certain measurements.

Definition 2.2.5: A **feature** f is a fuzzy set $f : \mathfrak{S} \to [0,1]$ on $\mathfrak{S}$. We say that a stroke **has feature** f if $f(s) = 1$ and that a stroke **does not have feature** f if $f(s) = 0$. △

So, features are the same objects as the elements of a sample dispersal: functions into $[0,1]$. The difference is the point of view: For any given line a user wants to write on a touch surface they have a respective set of good examples in mind — the super-level set $\{s \in \mathfrak{S} \mid m_l(s) \geq \lambda_l\}$ from Definition 2.2.4 — and then they try to produce one of these good examples on the touchscreen. So, the hand of the user, which can be seen as a random variable, follows loosely the distribution indicated by the sample dispersal. Features, in contrast, are values that get computed for a recorded stroke and from which we want to infer which line it supposedly represents.

If we have a set of features $f_1, ..., f_m$ we sometimes consider it just as a collection; sometimes we consider the function

$$F = \begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix} : \mathfrak{S} \to [0,1]^m.$$

The latter we call a **feature vector**.

Note that both features and feature vectors introduced here are used in the same way as in general Machine Learning. However, we want to think of feature not just as any function from the abstract object space $\mathfrak{S}$ into the numerical space $[0,1]$, but as concrete geometrical properties of strokes. To illustrate this, we give such a feature concretely, and it will serve as the go-to example for the rest of this thesis.

Example 2.2.6: Define functions $\textsc{Length} : \mathfrak{S} \to \mathbb{R}$ as

$$\textsc{Length}(s) := \sum_{i=1}^{n-1} \|P_{i+1} - P_i\|_2$$

and $\textsc{Straightness} : \mathfrak{S} \to [0,1]$ as

$$\textsc{Straightness}(s) := \frac{\|P_1 - P_n\|_2}{\textsc{Length}(s)}.$$

The latter one only takes values in the interval $[0,1]$, due to the triangle inequality. On a more descriptive level: The numerator is the distance between start and end point of the stroke — a quantity describing the extensiveness of the stroke — and the denominator is the length of the stroke — describing the complexity of the stroke. And this makes it also graphically clear that the image of L lies in the interval $[0,1]$. Furthermore, we have $\textsc{Straigthness}(s) = 1$ if and only if s is a straight line and $\textsc{Straigthness}(s) = 0$ if and only if s is a closed loop; i.e. when start and end point coincide. Because of that, we call this feature the straightness of a stroke.

In general, the connection between different feature values can be very, very complicated. But straight lines are unique up to similarity maps. So, for many practical cases, we can deduce all feature values from the fact that $\textsc{Straigthness}(s) = 1$; or often even when it is close to 1.

Conversely, if $\textsc{Straigthness}(s) = 0$ or at least very small, we might have to analyse the stroke separately. Again using similarity maps, we often normalise strokes by fixing their start, and end point at certain positions — briefly discussed above after introducing stroke spaces — which often leads to degenerate cases when they coincide or are close to each other. $\triangle$

The above description of features as fuzzy sets is enough to describe any feature-based handwriting recognition algorithm adequately: The real-valued functions that are used to compute geometric properties of strokes are often naturally bounded; especially when strokes are normalised into $\mathfrak{Q}$. Hence, any such function can be normalised to map into $[0,1]$. There are, however, two reservations to this.

First, the largest function value we obtain might not be the guarantee for the presence of the geometric property we want, and the lowest value might not indicate its absence. Thus, simply rescaling the image range $[a, b]$ to $[0, 1]$ via $x \mapsto \frac{x-a}{b-a}$ might collide with the standard interpretation of fuzzy sets. Second, there might be more than one feature value describing the desired property. We introduce the following notion to allow for a little bit more flexibility in handling these cases and to retain more information from the primal computations.

Definition 2.2.7: We call a function $f : \mathfrak{S} \to [-1, 1]$ a **proto-feature**. Any such proto-feature induces a feature on $\mathfrak{S}$ in two canonical ways. First, the **shifted** feature $f_{/} : \mathfrak{S} \to [0, 1]$ given by

$$f_{/}(s) := \frac{1}{2} \cdot f(s) + \frac{1}{2}$$

and the **folded** feature $f_{\vee} : \mathfrak{S} \to [0, 1]$ given by

$$f_{\vee}(s) := |f(s)| \, .$$

$\triangle$

The indices of these induced maps visualise the graphs of the transformations maps $x \mapsto \frac{1}{2}x + \frac{1}{2}$ and $x \mapsto |x|$, respectively. The reason we introduce this additional term is that in practice many functional terms for geometric properties emerge such that the image will be $[-1, 1]$ as we will see in Section 2.3.3. Furthermore, this allows for a more nuanced interpretation of these functional values. We will see this in the next example. Before that, we present the general idea: Note that both induced features take on the value 1 if the original proto-feature takes value 1. So, the interpretation for a proto-feature taking the value 1 will be the same as with ordinary features: that the stroke in question "has" the geometric property the proto-feature measures. The difference is in strokes s with $f(s) = -1$.

When we shift a proto-feature f we interpret $f(s) = -1$ as the stroke not having the property in question. I.e., we want $f_{/}(s) = 0$. When we fold a proto-feature we see this property as taking shape in two different polar opposite ways indicated by $f(s) = 1$ and $f(s) = -1$, respectively. The value $f(s) = 0$ is then the one representing the absence of the property, since $f_\vee(s) = 0$.

Example 2.2.8: For any stroke $s \in (\mathbb{R}^2)^n$ such that $\textsc{Straightness}(s) < 1$ consider the function

$$f : s \mapsto \frac{\sum_{i=2}^{n-1} [\hat{P}_1, \hat{P}_i, \hat{P}_n]}{\sum_{i=2}^{n-1} \left|[\hat{P}_1, \hat{P}_i, \hat{P}_n]\right|}.$$

As mentioned in Lemma 2.1.5, the determinants above are, up to a factor of $\frac{1}{2}$, the areas of the triangles $P_1P_iP_n$. The condition $\textsc{Straightness}(s) < 1$ makes sure that not all points P_i lie on the start-end line $P_1 \vee P_n$. I.e., the denominator is not zero.[9]

With the triangle areas in mind, it is easy to see that f takes only values between -1 and 1 which makes it a proto-feature. Moreover, $f(s) = 1$ if and only if all points of s lie on the right-hand side of the line $P_1 \vee P_n$ (when oriented from start to end point) and $f(s) = -1$ if all points lie on the left. This means that the shifted feature $f_/$ describes the geometric property

All points lie on the right-hand side of the start-end line

whereas the folded feature $f_\vee$ describes

All points lie on one side of the start-end line.

$\triangle$

We can transform proto-features into ordinary features in many different ways. Most reasonable ones are, however, variations of the shift and fold introduced in Definition 2.2.7. E.g., we can consider

$$f_+(s) := \max(0, f(s))$$

instead of $f_/$ and

$$f_2(s) := f(s)^2$$

instead of $f_\vee$. The first, f_+, simply cuts off negative values and equates to the linear rectifier used as an activation function in neural network design. The

[9]In praxis, it is advisable to demand $\textsc{Straightness}(s) < 1 - \varepsilon$ for a suitable threshold $\varepsilon > 0$ to guarantee numerical stability.

second, f_2, is a smooth alternative to $f_\vee$ that, however, overemphasises values near 0. Choosing one way or another to modulate proto-feature depends, of course, on the actual geometric properties in question. But as a general heuristic, we will use shifting and folding as they create minimal disparities between proto-features and features.

Finally, any machine learning application needs some training data. Our goal for this thesis, as was mentioned already in Chapter 1, is to have a small set of such data.

Definition 2.2.9: Let $\mathbb{A}$ be an alphabet, $\mathfrak{T}$ a collection of good samples for $\mathbb{A}$ and $f_1, ..., f_m$ a set of features. Then we define the **training context** $\mathcal{T}$ of $(\mathbb{A}, \mathfrak{T}, \{f_1, ..., f_m\})$ to be the multi-valued context

$$\mathcal{T} = \left(\bigcup_{l \in \mathbb{A}} \mathfrak{T}_l,\ \ \mathbb{A} \cup \{f_1, ..., f_m\},\ \ [0,1],\ \ I \right).$$

The object set is the union of all sets of good samples for all stroke types in the alphabet and the attribute set is the union of the alphabet and the feature set. We then build the incidence relation I in the following way:

— For all strokes $s \in \bigcup_{k \in \mathbb{A}} \mathfrak{T}_k$ and all stroke types $l \in \mathbb{A}$ we add $(s, l, 1)$ to I if $s \in \mathfrak{T}_l$ and we add $(s, l, 0)$ if $s \notin \mathfrak{T}_l$.

— For all strokes $s \in \bigcup_{k \in \mathbb{A}} \mathfrak{T}_k$ and all features f_i we add $(s, f_i, f_i(s))$ to I.

△

So, we simply note for every sample stroke which type it represents and which feature values it has and write this information into a table.

To examine how much features of strokes are affected by transforming the strokes, we apply a straightforward, well-known statistical method: the Kolmogorov-Smirnov test. We will use it as a multi-purpose tool for certain applications, but any other reasonable statistical test can be used. In particular, if there is additional information on the distributions involved.

In general, the Kolmogorov-Smirnov checks whether a random variable has a given distribution or whether two random variables have the same distribution. Also, this is determined based on finitely many samples from the random variables. For us this means the following: We can consider features as random variables and compare their values for a given set of strokes before and after we apply a geometric transformation. Before we get into the application of this to strokes and feature we give the general statement.

Proposition 2.2.10: *Let X and Y be two random variables and $x_1, ..., x_A$ be samples for X and $y_1, ..., y_B$ be samples for Y. Then let $F_{X,A}$ and $F_{Y,B}$ be the empirical distribution functions of X and Y, respectively. I.e.,*

$$F_{X,A} : \mathbb{R} \to [0,1], \quad x \mapsto \frac{1}{A} \sum_{i=1}^{A} \mathbb{1}_{x_i \leq x}$$

and analogously for $F_{Y,B}$. Here, $\mathbb{1}$ describes the indicator function of subsets of $\mathbb{R}$. Then we consider the null hypothesis:

X and Y are equal in distribution.

Given a level of significance α, the null hypothesis is rejected if

$$D_{A,B} := \sup_{x \in \mathbb{R}} |F_{X,A}(x) - F_{Y,B}(x)| \quad > \quad \sqrt{-\frac{1}{2} \cdot \frac{A+B}{AB} \cdot \ln(\alpha)}.$$

Intuitively this makes sense: Given two "good" approximations for the cumulative distribution function of X and Y, like the empirical distribution functions are, the random variables cannot have the same distributions if their approximations differ too much.

When using this to analyse features and strokes, we equip both $\mathfrak{S} = \left(\mathbb{R}^2\right)^n$ and $[0,1]$ with the Borel σ-algebra such that they become measurable spaces. And since most features we work with are continuous, they are therefore directly measurable function. Then we usually consider one of two cases:

— Two finite sets $S, T \subset \mathfrak{S}$ and a feature f with which we then compare $f(S)$ and $f(T)$ as the samples in the Kolmogorov-Smirnov test. This allows us to see whether samples for different stroke types behave differently with respect to a certain feature.

— One set $S \subset \mathfrak{S}$ and two features f_1, f_2 in order to compare $f_1(S)$ and $f_2(S)$. With this we can analyse whether two different features behave differently for one or several stroke types.

The second part is particularly handy to compare different function terms for the same geometric property. For example, consider

$$\textsc{Straightness} : \mathfrak{S} \to [0,1], \qquad s \mapsto \frac{\|P_n - P_1\|_2}{\sum_{i=1}^{n-1} \|P_{i+1} - P_i\|_2}$$

again. The essential insight is that this feature takes the value 1 if and only if the stroke in question is a straight line segment. But this can be computed differently. With the determinant notation from Section 2.1.1 and using Lemma 2.1.5, we can also consider

$$s \mapsto 1 - \frac{1}{n-2} \sum_{i=2}^{n-1} \left[\hat{P}_1, \hat{P}_i, \hat{P}_n\right]^2 .$$

This then only makes sense if we define this feature on a normalised space such as $\mathfrak{Q}$ in order to assure that the triangle areas represented by $\left[\hat{P}_1, \hat{P}_i, \hat{P}_n\right]$ are bound from above by 1.

The interpretation of the feature value 1 is here the same as with the fraction of distances introduced before: the stroke has to be a straight line segment. But due to the writing/recording process, we almost always have to work with values less than 1. The Kolmogorov-Smirnov test allows us the see any significant differences in two such function terms with respect to the training data.

As stated above, we use this test in examples and applications to get indications how sets of feature values might or might not be connected.

To illustrate how all the definitions in the section work together, we now give an overview of the handwriting recognition algorithm used in the ALICE iBook.

2.3 An overview of ALICE:HWR

This section here serves as an overview of the base ideas of ALICE:HWR. Some things will be clearer after Chapters 3 and 4, so we will return to a discussion of ALICE:HWR in Chapter 6.

2.3.1 The base structure

ALICE:HWR works with the stroke types

$$\begin{aligned}\mathbb{A} = \{&\texttt{1a},\ \texttt{1b},\ \texttt{2},\ \texttt{3},\ \texttt{4a},\\ &\texttt{4b},\ \texttt{5a},\ \texttt{5b},\ \texttt{6},\ \texttt{7},\\ &\texttt{8a},\ \texttt{8ar},\ \texttt{8b},\ \texttt{9a},\ \texttt{9b},\\ &\texttt{0a},\ \texttt{0ar},\ \texttt{0b},\ \texttt{-}\}.\end{aligned}$$

Written nicely, they are supposed to look like the strokes seen in Figure 2.9. The difference between `8a` and `8ar`, and `0a` and `0ar` is that the `r`-version is written backwards. Checking whether a stroke was written backwards is relatively simple and was done in ALICE:HWR, see Appendix B, but for these two types it turned out to more effective to hard-code the reversed stroke types.

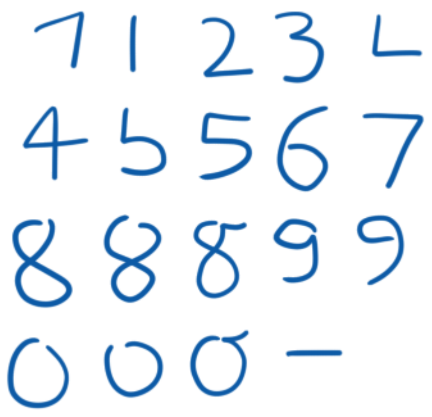


Figure 2.9: *(Examples for) Stroke types used in ALICE:HWR.*

Moreover, we use a list of 25 features which are listed in Section 2.3.3 below.

Based on them, the handwriting algorithm is set up and trained in the following way:

1. A small set of model strokes — i.e., good samples — is used to represent each stroke type.
2. Based on various feature values, certain rules are determined when a recorded stroke does *not* represent one of the types. Formal Concept Analysis is used to compute these rules, and this step is explained in Section 4.2.
3. The number of samples is increased using the geometric transformations presented in Chapter 3.
4. An idealised feature vector for every stroke type is calculated based on the feature vectors of the multiplied samples.

Afterwards, when a stroke r is recorded it gets classified using the following steps; illustrated in Figure 2.10:

1. Using the decision rules established beforehand, the strokes types that r cannot represent are excluded from $\mathbb{A}$.
2. The feature vector of r is compared to the idealised stroke type feature vectors from before via 1-nearest neighbour classification. The method to compare these feature vectors is explained in Section 4.3.

All recorded strokes — both during the training and the classification phase — are pre-processed using the methods shown in Section 2.3.2. Many of these steps are based on the notions and results of the subsequent chapters. Because of that, we will revisit this algorithm in Chapter 6 again after explaining them.

One point we do want to emphasise here already: The features used in ALICE:HWR are *theoretical-based* and *constructive*. That means, they are chosen such that they *should* work in theory and *do* work for many examples in

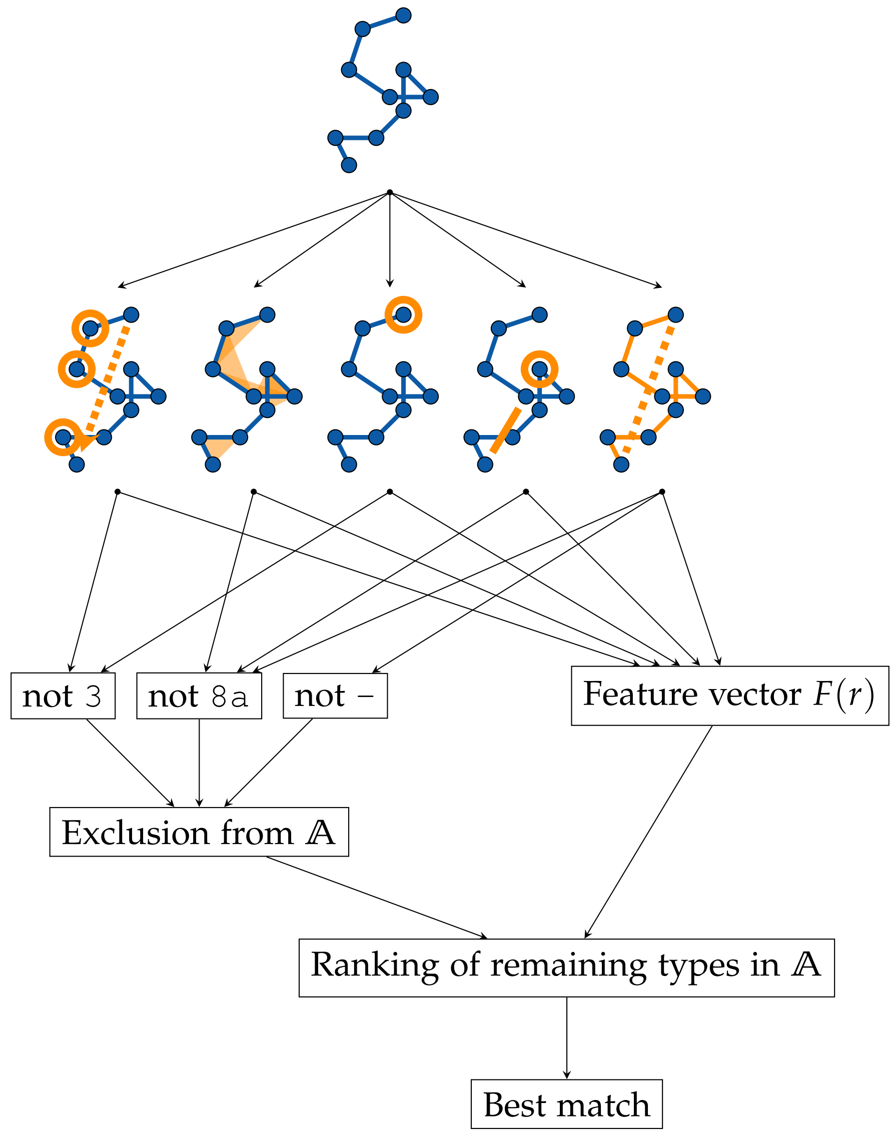

not 3
not 8a
not –
Feature vector $F(r)$
Exclusion from $\mathbb{A}$
Ranking of remaining types in $\mathbb{A}$
Best match

Figure 2.10: *Classification in ALICE:HWR.*

praxis.[10] So, implementing them basically means that the developer tells the algorithm what to look for. The decision rules found via Formal Concept Analysis are then produced automatically though; and they describe explicitly which geometric properties do — or rather do not — characterise the stroke types in question. Because of the small sample size used, however, they have to be adjust manually to work in practice. This results in rules like the following:

A stroke cannot represent the type `1a` *if its start and end point are too close together or it contains too many left turns.*

[10]See [11] for more information.

Finding this kind of classification rules is the motivation of each chapter in this thesis. However, most statements will focus on individual parts and the connexion to this goal might not always be clear. Again, we refer to Chapter 6 in which we look back at ALICE:HWR as a whole.

The rest of this section will present the central ideas of the pre-processing step and all features used in ALICE:HWR.

2.3.2 Pre-processing

Here we will present some aspects of how strokes were recorded and pre-processed before the actual analysis.

The actual recording from the touchscreen depends foremost on the sample rate of the device. In ALICE we used iPads — to be precise the models iPad 5 and iPad Air 2 — and Apple's *iOS Device Compatibility Reference* (see [1]) tells us that their sample rate is 60Hz. There is a difference between the sample rate and the delivery rate though: The former is the number of times the operating system processes the touchscreen data; the latter is the number of times the operating system provides these data to other programs. As the delivery rate is 60Hz for all Apple devices, we assume that it is equal to the sample rate. So, with this sample rate, we can record 60 points per second.[11]

As we mentioned in the context of stroke spaces in Section 2.2, it is desirable to have a minimal distance between points. At various points in the HWR process we might divide by the distance between consecutive points and to assure that this computation is numerically stable (or at least stable enough) it is advisable to guarantee a lower bound for these distances. In particular, ALICE uses *CindyJS* which performs up to 1000 ticks per second.[12] This means that up to 16 consecutive points end up with the same coordinates. To avert this, we introduce an $\varepsilon > 0$ and do the following.

[11]Modern Apple devices allow for much higher precision using so-called *coalesced touches*. This function allows the software to use up to 240 touch events. For the purposes in ALICE — for handwriting recognition or otherwise — the standard of 60Hz is more than enough.

[12]The actual number fluctuates a lot depending on the device used and the total number and complexity of operations executed.

```
if $\|(lastpoint) - (fingerposition)\|_2 > \varepsilon$ then
    Record $(fingerposition)$ as a new point.
```

This now assures that consecutive points are not too close together and it also limits the number of points necessary to store.

The next step is usually to take these raw data points and to smooth out errors and noise. These often emerge via the technical limitations of both the touch surface itself and the computer system used and also via the "jittery" finger movement of a user — especially when writing slowly.

The most straightforward way to reduce these errors is to apply a Gaussian filter to every stroke. This a standard tool in graphics processing and we will analyse it in the next chapter. As we will show in Theorem 3.1.18, applying a Gaussian filter over and over again will eventually erase "everything" from a stroke until only a straight line remains. Applying it just a few times, however, eliminates noise and errors. This can be observed in Widget A.4 in the companion iBook.

The last step in the recording process is the re-sampling of the stroke. We assume that, after smoothing, all the recorded points lie on the continuous curve the user wanted to draw. **Re-sampling** now finds other/more points on this curve that are, in one way or another, better.

The first goal of this is to reduce the number of points to discard unnecessary information immediately. The coordinates of consecutive points on a stroke are correlated, as the speed at which a user moves their finger/stylus while writing is not only bounded but also relatively low (with respect to the sampling frequency).

The second goal is to generate equidistant points along the stroke. This is desirable as many features are curvature-sensitive. The definition of curvature for smooth curves $\gamma : [0,1] \to \mathbb{R}^2$ makes only sense if they are parametrised with respect to the arc length — i.e., if $\|\gamma'\|_2 = 1$. The discrete analogue for a stroke $(P_i)_{i=1}^n$ is that $\|P_{i+1} - P_i\|_2$ is constant for all i.

A popular method to re-sample strokes is to use cubic splines: We can look at two points $B, C \in \mathbb{R}^2$ and connect them with a cubic curve. I.e., we want a function $p : [0,1] \to \mathbb{R}^2$ with $p(0) = B$ and $p(1) = C$ and we want p to be a polynomial function (with coefficients in $\mathbb{R}^2$) of degree 3. Similar to Bézier splines, we introduce auxiliary points A and D to guide the shape of the curve. When we demand that the tangent at the start should point from A to C and at the end from B to D, we get a special **cubic Hermite spline**.[13]

Lemma 2.3.1: *Let A, B, C, D be four points in $\mathbb{R}^2$ and let $\alpha \in [0,1]$ be a form parameter. Consider a cubic polynomial curve $p_\alpha : [0,1] \to \mathbb{R}^2$ such that*

$$\begin{aligned}
p_\alpha(0) &= B,\\
p_\alpha(1) &= C,\\
p'_\alpha(0) &= \alpha \cdot (C - A),\\
p'_\alpha(1) &= \alpha \cdot (D - B).
\end{aligned}$$

Then, p_α is explicitly given by

$$p_\alpha(t) = \begin{pmatrix} A & B & C & D \end{pmatrix} \cdot \begin{pmatrix} 0 & -\alpha & 2\alpha & -\alpha \\ 1 & 0 & \alpha - 3 & 2 - \alpha \\ 0 & \alpha & 3 - 2\alpha & \alpha - 2 \\ 0 & 0 & -\alpha & \alpha \end{pmatrix} \cdot \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}.$$

Idea of proof. p_α has to be of the form

$$p_\alpha(t) = \Lambda_3 t^3 + \Lambda_2 t^2 + \Lambda_1 t + \Lambda_0$$

with $\Lambda_3, \Lambda_2, \Lambda_1, \Lambda_0 \in \mathbb{R}^2$. The four equations describing p_α are linear in the Λ_i, so it is a linear system which ends in the given result. □

In order to re-sample a stroke, we look at consecutive points $P_i, P_{i+1}, P_{i+2}, P_{i+3}$ and interpolate between P_{i+1} and P_{i+2} with P_i and P_{i+3} being the auxiliary points. In order to interpolate between the first and last pair of points, we extrapolate the stroke by the “best guess”: We simply add the vector

[13]General Hermite splines are represented by specifying the tangents at start and end. Here we have a special case as the tangents are given by specific control points.

$P_1 - P_2$ to the start point P_1 to get an additional auxiliary point P_0; and analogously another auxiliary point P_{n+1} at the end of the stroke. At the end of the process, they are discarded again.

The final consideration is that we want an equidistant sampling of the resulting cubic spline. We do not want to compute the actual length of it though. As the original stroke will have its points very close to each other, we use the piecewise linear curve through them as a proxy and estimate the length of the spline via the LENGTH function introduced in Example 2.2.6, which adds up the distances of consecutive point pairs. With that in mind, we get the following re-sampling algorithm.

Algorithm 2.3.1: RE-SAMPLING

```
Input  : A stroke $s \in \left(\mathbb{R}^2\right)^m$, a form parameter $\alpha = [0,1]$ and a sample
         rate $n \geq 2$.
Output: A stroke $\tilde{s} \in \left(\mathbb{R}^2\right)^n$.
1 Set $P_0 = 2 \cdot P_1 - P_2$ and $P_{m+1} = 2 \cdot P_m - P_{m-1}$.
2 Start with the stroke $\tilde{s} \leftarrow (P_1)$.
3 for $i = 2, ..., n-1$ do
4     Find the integer $1 < j < m-1$ such that
      $\frac{\text{LENGTH}\left((P_1,...,P_{j-1})\right)}{\text{LENGTH}(s)} \leq \frac{i-1}{n-1} < \frac{\text{LENGTH}\left((P_1,...,P_j)\right)}{\text{LENGTH}(s)}$.
5     Set $t \leftarrow \left(\frac{i-1}{n-1} - \frac{\text{LENGTH}\left((P_1,...,P_{j-1})\right)}{\text{LENGTH}(s)}\right) \cdot \frac{\text{LENGTH}(s)}{\left\|P_j - P_{j-1}\right\|_2}$
6     Take the cubic curve $p_\alpha$ from Lemma 2.3.1 between the points
      $P_{j-1}, P_j, P_{j+1}, P_{j+2}$ and compute the new point $Q_i \leftarrow p_\alpha(t)$.
7     Set $\tilde{s} \leftarrow \tilde{s} + Q_i$
8 Return $\tilde{s} + P_m$.
```

In ALICE:HWR the form parameter $\alpha = \frac{1}{2}$ is chosen and leads to **Catmull-Rom splines**. They were introduced differently in [6], but as explained in [64] they can be represented as an Hermite spline as introduced in Lemma 2.3.1 above if $\alpha = \frac{1}{2}$. Note that these splines are **uniform** Catmull-Rom splines and therefore not necessarily optimal for interpolation. The better alternative is given by centripetal Catmull-Rom splines as explained in [72]. But as the points on the originally recorded strokes are close together, we do not run the risk of creating cusps and loops with this interpolation. Moreover, we use this method to reduce the number of sampling points instead of rendering the whole spline (which would mean to increase the number of sampling points drastically). So, seeing small loops created in the process is unlikely.

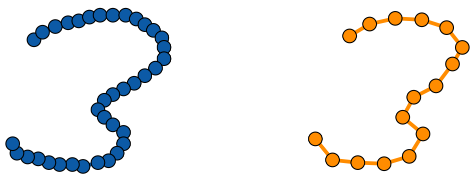


Figure 2.11: *Equidistant re-sampling of a stroke via Catmull-Rom splines.*

As we sample the continuous curve which runs through the points of the original stroke, we have to make sure that these points lie on a “nice” curve. I.e. if they zig-zag too much, so will the resulting stroke — at least when n is very large. So, we use this re-sampling only after applying a Gaussian filter to smooth the original stroke.

The above steps — error reduction and re-sampling — are relevant in any application of data analysis in one way or another; even though our methods are built explicitly for strokes. There is, however, one particular problem that only emerges in HWR in this form: hooks.[14]. They are minuscule line fragments at the start and end of strokes, and they appear during “ordinary” writing on paper, too. They arise from the pen tip touching the surface too early and leaving it too late. Hence they point towards the direction the pen tip moves before/after the stroke.

The problem is that many features used to describe strokes get distorted by these hooks — most notably orientation features describing the position and direction the first and last parts of a stroke and also features processing local curvatures. To remove these hooks, many different authors came up with many different solutions. A popular one can be found in [69]. Most of them use the fact that hooks can be characterised by, first, their position and, second, their

[14]Sometimes called hooklets.

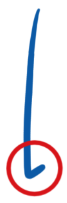


Figure 2.12: *A hook at the end of a stroke.*

sudden change in (local) curvature or, in other words, by the angles between consecutive sampling points. This change is then often processed into a single scalar value, i.e., a feature h. If this value turns out to be larger than a certain bound, the points at the end of the stroke get cut off or smoothed out in a certain way.

This idea is undeniably sensible, but it depends on the design of the feature h, the threshold it has to overshoot and the subsequent alteration of the points. Here we will propose an alternative which does not check whether hooks are present and still eliminates them.

The basic idea will be that

the "relevant" information/shape of a stroke is located in the middle.

With this, we justify that we can just cut off both ends of a stroke without losing any information. The actual algorithm will be a little bit more intricate, but first, we want to see how reasonable this is. In Figure 2.13 we see 18 different strokes (without hooks) sampled by 16 points each. Then we cut off the first and last two points; i.e., 25% of the stroke in total. We can see that most of them are still readable.

The main problem is observable in the third row: When the ends are part of a loop — either both as in the `8` and `0` or just one of them as in the `9` — this loop gets "cut open". This might make the stroke unreadable or turns it into another stroke. (`0` into `U` in the figure.) But overall, many strokes stay recognisable.

If there is a hook in the last, say, 10% of a stroke and we cut off these points, the hook will be removed. Or in other words: we define a hook to be something that happens in the first (or last) 10% of a stroke. But as illustrated in Figure 2.13, we cannot assume that all strokes keep their recognisable shape by this process. So, we add points back to the stroke (in order to preserve this shape) such that they do not form a hook.

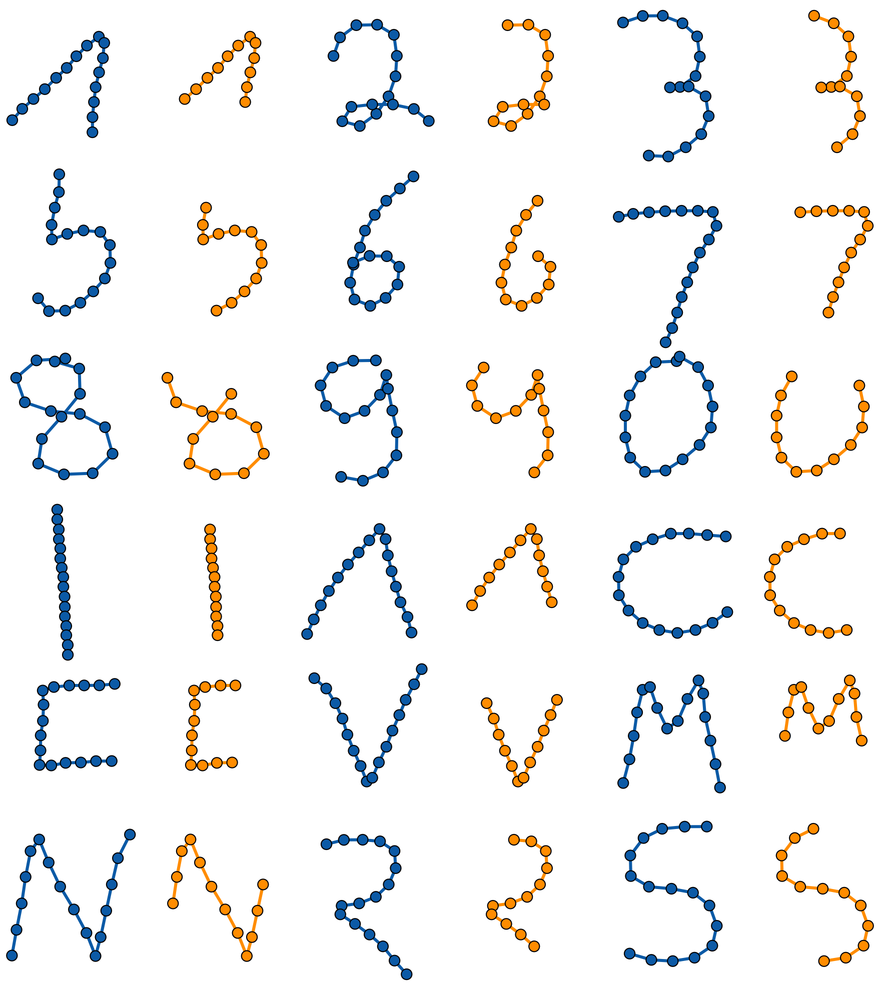


Figure 2.13: *Cutting off the ends of strokes.*

Let A, B, C be the last three points of a stroke. We want to extrapolate the stroke by taking the angle of the stroke at B and recreate it at C. Denote the vectors $v := B - A$ and $w := C - A$ and let $\alpha := \frac{1}{2}\measuredangle(v, w)$ seen as a signed angle. We use half of the actual angle at B in order to simulate a fade-out motion by the user; assuming they write "nicely". Let R be the rotation by α and let D be the extrapolated next point on the stroke. Then we set

$$D = C + Rw.$$

In ALICE:HWR, the strokes are (almost) equidistantly re-sampled, but in general this might not be the case. For the extrapolation of strokes based on angles this means that we have to re-scale the step we make based on the progression form A to B to C. I.e., we set

$$D = C + \frac{\|w\|_2}{\|v\|_2} \cdot Rw.$$

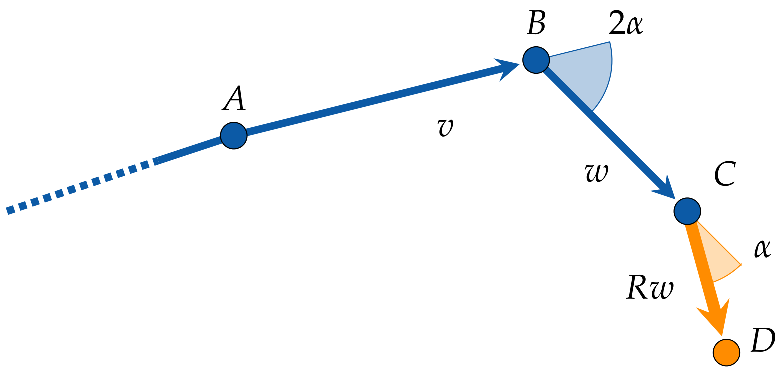


Figure 2.14: *How to extrapolate the end of a stroke.*

How this turns out is visualised in Figure 2.15. There, the stroke is extrapolated twice via the given method above. Note, that when this is done multiple times, the result of the previous extrapolations is used in the subsequent one. As we take half the previous angle at every step, the extrapolation becomes flatter. However, if the points of the initial stroke are not equidistant, the extrapolation points might move much farther away than intended.

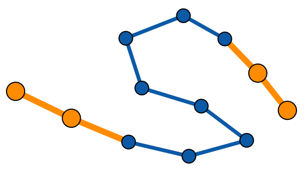


Figure 2.15: *Extrapolating the end of a stroke.*

For hook removal, we now have the two steps of cutting off the ends of strokes and then recreating them based on the angle between the remaining pieces. So far, this process is systematic and effective — it will definitely eliminate any hooks that might have been present. However, as seen in Figure

2.13, closed loops at the ends of strokes — e.g., in `8`, `9` and `0` — are affected a lot by the cutting-off step and might end up looking like another stroke type. These strokes depend a lot on the arch of the stroke to form these loops — extrapolating them with a smaller angle as done above will not re-build the loop.

To take this into account, we compute a convex combination of the initial stroke and the altered one and the make the coefficients dependent on the START LOOP of the stroke. It is equal to feature f_{18} from Section 2.3.3 below. It measures how close a point along the stroke comes to the start point. This describes whether there is a loop at the beginning of the stroke. It can be thought of as a local variant of the STRAIGHTNESS of a stroke.

Using this particular feature is a heuristic choice — any other fuzzy function that gives the likelihood that cutting-off the ends of a stroke destroys an important loop can be used, too. Therefore, we do not give the precise definition of START LOOP here and refer to Section 2.3.3 for that.

We will modulate it by a function $\mu_c : [0,1] \to [0,1]$ with $c \in \mathbb{R}^+$ which is given by

$$x \;\mapsto\; \frac{c \cdot x}{(c-1) \cdot x + 1}.$$

It allows for a controlled way to increase values in the interval $[0,1]$ by the factor c without leaving the interval. To see this, first observe that it fulfils the functional equation

$$1 - \mu_c(1-x) = \mu_{\frac{1}{c}}(x) \tag{2.1}$$

for all x. Additionally, it has slopes

$$\left.\frac{d}{dx}\mu_c\right|_{x=0} = c \qquad \text{and} \qquad \left.\frac{d}{dx}\mu_c\right|_{x=1} = \frac{1}{c}.$$

The first one can be interpreted as if the the function μ_c takes values close to zero and multiplies them with c. But it is not clear how something similar should work for values close to 1: When we interpreting the elements of $[0,1]$ as probabilities or fuzzy values, how should values close to 1 be multiplied by a number like 5? E.g., saying that one event is 5 times as likely as another if the second one occurs with probability 90% does not make sense straight away.

But the slope of μ_c at 1 together with the functional equation (2.1) means that for values x close to 1 the complement $1 - x$ gets multiplied by $\frac{1}{c}$. Of course this is not the definitive way to do it, but we can say:

Multiplying large fuzzy values x by c is the same as multiplying $1 - x$ by $\frac{1}{c}$.

This interpretation, for example, means that quintupling a probability of 90% should give 98%. And μ_c models this. It is a smooth bijection that respects this interpretation both for small and large values.

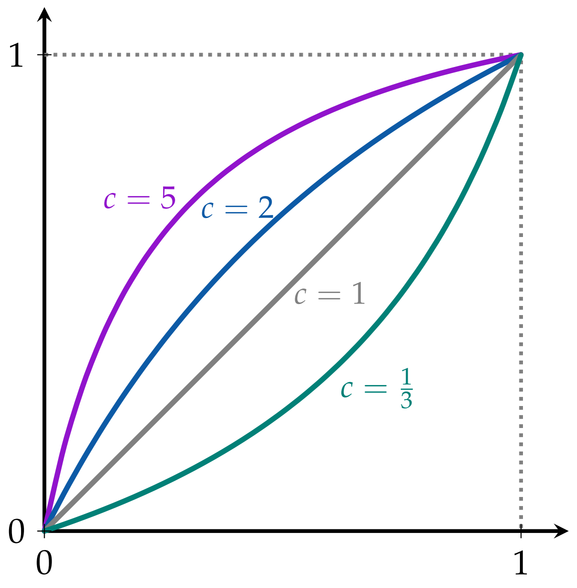


Figure 2.16: *Graphs of μ_c for various parameters c.*

This modulation function is used at various places in ALICE:HWR to increase or decrease fuzzy values. Here will we now us it to quintuple the feature value Start Loop(s) for a recorded stroke s. We get the following algorithm for hook removal (at the start of a stroke).

Algorithm 2.3.2: Hook Removal

```
  Input  : A stroke $s \in \left(\mathbb{R}^2\right)^n$.
  Output: A stroke $\tilde{s} \in \left(\mathbb{R}^2\right)^n$ without a hook at the start.
1 Cut off the first $\lfloor 0.1 \cdot n \rceil$ points at the start of $s$, to get a new stroke $c$.
2 Extrapolate $\lfloor 0.1 \cdot n \rceil$ times at the start of $c$ to get a new stroke $e$
    consisting of $n$ points.
3 Compute the convex combination
      $\tilde{s} = \mu_5 (\textsc{Start Loop}(s)) \cdot s + (1 - \mu_5 (\textsc{Start Loop}(s))) \cdot e.$
```

Of course, this algorithm should be applied to the end of the stroke, too, using End Loop instead of Start Loop. We do not claim that this algorithm is the best for removing hooks. But Figure 2.17 shows its viability.

Remark 2.3.2: We can, in fact, easily prove that Algorithm 2.3.2 works correctly. To do so, however, we need a formal definition of what a hook is. And if we make the reasonable assumptions that

1. a hook is something that happens in the first 10% of a stroke and
2. a hook is characterised by both large angles and high a variance among the angles of the line segments which comprise the stroke,

then the proof becomes trivial: As we cut off the first 10% of the stroke, any hook that was present will be removed. And since we use half of the last angle in the extrapolating step, the new part we attach will be flat and not form a hook.

To evaluate the last step of building a convex combination in Algorithm 2.3.2, however, we probably need something that encodes the 'size' of a hook and to use a more purposeful feature than Start Loop.

But as stated in the beginning, any such considerations hinge on the definition of the term 'hook'. △

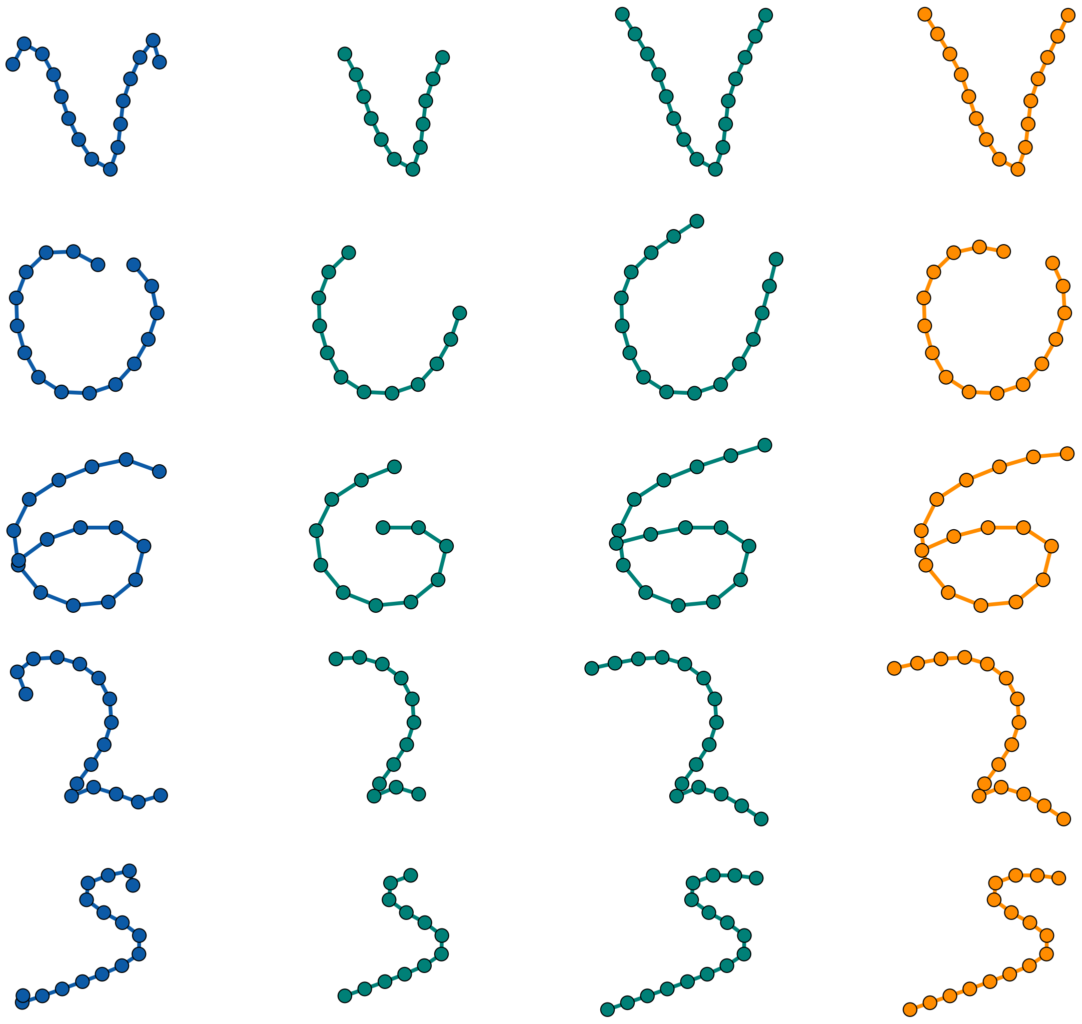


Figure 2.17: *Algorithm 2.3.2 applied to a selection of strokes. The first column are the original strokes. In the second column start and end parts have been cut off. In column three, they were extrapolated again. And the fourth column is a convex combination of the first and third.*

Combining all ideas for pre-processing presented above leads to Algorithm 2.3.3 below. In it, we use negative point indices. For a stroke (or general list or array) $r = (P_i)_{i=1}^{n}$ of length n we define its $(-k)$-th point/element as $P_{-k} := P_{n-k+1}$. In particular, the last point is the (-1)-st.

Algorithm 2.3.3: Pre-processing in ALICE:HWR

```
Input  : A stroke $r = (P_i)_{i=0}^{n} \in \left(\mathbb{R}^2\right)^{n+1}$ (for some indeterminate
         $n \in \mathbb{N}^+$).
Output: A stroke $r' \in \left(\mathbb{R}^2\right)^{24}$.
1 Set $r \leftarrow (P_1, ..., P_n)$.
2 if $n < 4$ then
3     Set $r \leftarrow (P_1,\quad \frac{3}{4}P_1 + \frac{1}{4}P_2,\quad \frac{3}{4}P_{-1} + \frac{1}{4}P_{-2},\quad P_n)$ and $n \leftarrow 4$.
4 for $i$=1, ..., $n$ do
5     Get uniformly distributed random numbers $\varepsilon, \delta \in [0, 0.001]$.
6     Set $P_i \leftarrow P_i + (\varepsilon, \delta)$.
7 Re-sample $r$ with 24 points using Algorithm 2.3.1.
8 Set $r \leftarrow \mathcal{G}^2_{A_{0.02}}(r)$.
9 Apply the Hook Removal Algorithm 2.3.2.
```

For the most part, this algorithm is just a concatenation of the algorithms presented before. Points worth mentioning, though, are: In line 1 we cut off the first point of the recorded stroke regardless of any concerns we talked about above in the context of hook removal. The reason is that strokes in the ALICE iBook are intended to be written with the finger of the user; not a stylus. And then it turned out that the very first point of the stroke has so much variance in its relation to the rest of the stroke, that it is much simpler and more effective to exclude it every time.

Lines 2 to 4 guarantee that the stroke has at least four points which is necessary for the other algorithms to work properly. And lines 5 and 6 introduce a little bit of noise into the stroke as a precaution, since Algorithm 2.3.1 does not perform well on straight line segments.

Lastly, we smooth out the stroke in line 8 via a Gaussian filter. But as already said, this will be explained and illustrated in the next chapter.

The pre-processing steps and techniques presented are primarily used to make the features in the next section more stable and meaningful. In particular, the features are designed with a regular point distribution along the strokes in mind.

2.3.3 A list of all ALICE:HWR features

Below we give a list of all 25 features used in ALICE:HWR. We give them in the order they were implemented. As mentioned already above in Section 1.3.2, many of these features are taken from [11] and they are marked as such.

Recall how strokes look after the pre-processing phase: They have been equidistantly re-sampled with $n = 24$ points and then were smoothed using a Gaussian filter (see Section 3.1.4 for more information on that). In particular, the resulting strokes are not exactly equidistantly sampled anymore. Lastly, the hook removal algorithm is utilised such that the start and end parts are almost straight.

Afterwards, the stroke is normalised such that its bounding box is the square $[-1, 1]^2$ — so here we work in $2\mathfrak{Q}$. This is done by moving the stroke such that the centre of its bounding box is the origin and then the x-coordinates of the stroke are divided by the width of the bounding box and the y-coordinates by the height. When the width or the height is below a certain threshold, it is set to an "arbitrary large" default value in order to, firstly, not divide by (almost) zero and, secondly, preserve horizontal and vertical straight lines as straight lines.[15]

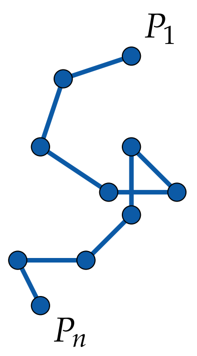


Figure 2.18: *The test stroke for illustrating the features.*

[15]In ALICE:HWR, this "large" value is 10.

When we now compute specific geometric properties, they often emerge as values in the interval $[-1, 1]$ — i.e., as proto-features — but not always. Here we will state just the function term/heuristic used to compute each feature. As explained above, all proto-features are shifted in ALICE:HWR and we assume that this re-scaling is done implicitly whenever necessary. As a reminder, we use the notation that any stroke s consists of points $P_1, ..., P_n$ and that any point P_i has the coordinates (x_i, y_i).

Additionally to the function term, we will give a short description of its geometric meaning and, sometimes, the reason why a particular feature was added. Moreover, every feature is illustrated qualitatively in an image; using the stroke shown in Figure 2.18. It is written from top to bottom and it is sampled with 10 points. This is much less than the 24 points used in ALICE:HWR, however, this makes it easier to visualise and see how the features are built.

START and END DIRECTION: This is the direction of the first and last part of the stroke — basically the discretised derivation of the stroke. We use the vector from the first to the third point (at each end). This makes it numerically more stable than going from the first to the second point, but is only sensible when the sampling rate n is large enough. Figure 2.19 depicts the "correct" conception using the first and second point.

$$\begin{aligned} f_1(s) &:= \frac{x_3 - x_1}{\|P_3 - P_1\|_2}, \\ f_2(s) &:= \frac{y_3 - y_1}{\|P_3 - P_1\|_2}, \\ f_3(s) &:= \frac{x_n - x_{n-2}}{\|P_n - P_{n-2}\|_2}, \\ f_4(s) &:= \frac{y_n - y_{n-2}}{\|P_n - P_{n-2}\|_2}. \end{aligned}$$

STROKE RETURN: This feature describes whether the start and end part of the stroke point towards the same direction, as in a 3, or not, as in a 2. So, it is the scalar product of START and END DIRECTION. In particular, it uses the third point instead of the second one, too.

$$f_5(s) := \frac{(P_3 - P_1) \cdot (P_{n-2} - P_n)}{\|P_3 - P_1\|_2 \cdot \|P_{n-2} - P_n\|_2}.$$

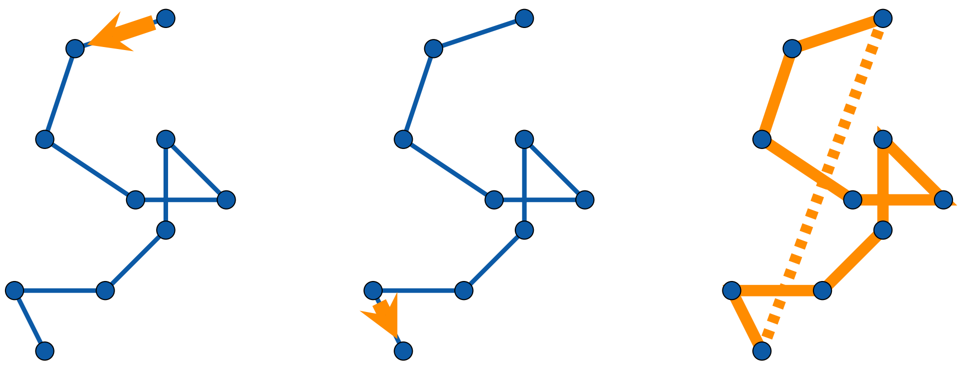


Figure 2.19: *Portrayal of:* Start Direction $(f_1, f_2)^T$ *(left),* End Direction $(f_3, f_4)^T$ *(centre) and* Straightness f_6 *(right).*

Straightness ([11]): How much the stroke is a straight line. This was used as an example above already. It compares the distance between start and end point and the length of the stroke. Hence it is 1 if and only if the stroke is a straight line.

$$f_6(s) := \frac{\|P_n - P_1\|_2}{\sum_{i=1}^{n-1} \|P_{i+1} - P_i\|_2}.$$

Loopyness ([11]): The fuzzy complement of Straightness—equalling 1 when the stroke is a loop, i.e., when start and end point coincide.

$$f_7(s) := 1 - f_6(s).$$

Left and Right Curvature: This computes the relative amount of points at which the local curvature is positive and negative. The curvature at any point P_i will be approximated by the signed distance from P_i to the line $P_{i-2}P_{i+2}$ for all $i = 3, ..., n-2$. (It is set positive if and only if P_i is to the left of the line.) Denote this distance as d_i and set

$$D := \{i \in \{3, ..., n-2\} \mid |d_i| > 0.03\}.$$

Here we ignore "curvatures" that are too small as they are most-likely errors or fluctuations in an actual straight part of the stroke. Then we set

$$L := \{i \in D \mid d_i < 0\} \quad \text{and} \quad R := \{i \in D \mid d_i > 0\}$$

to get

$$f_8(s) := \frac{|L|}{|D|},$$
$$f_9(s) := \frac{|R|}{|D|}.$$

If D is empty, set $f_8(s) = f_9(s) = \frac{1}{2}$.

See Chapter 5 for more on how to measure the curvature of strokes. As illustrated in Figure 2.20, we would like to compute this using the distance from P_i to $P_{i-1}P_{i+1}$. The version presented above gave higher numerical stability and better results in praxis. Since the sampling rate in ALICE:HWR is relatively high, there is no qualitative difference between using $i \pm 2$ and $i \pm 1$.

Left and Right of Start-End Line: This features computes the relative amount of points that lie on the left- and right-hand side of the start-end line. Its computation is basically identical to features 8 and 9 above. For all $i = 2, \ldots, n-1$ let d_i be the signed distance of P_i to the line P_1P_n. Then set

$$D := \{i \in \{2, \ldots, n-1\} \mid |d_i| > 0.1\}$$

and

$$L := \{i \in D \mid d_i > 0\} \quad \text{and} \quad R := \{i \in D \mid d_i < 0\}$$

to obtain

$$f_{10}(s) := \frac{|L|}{|D|},$$
$$f_{11}(s) := \frac{|R|}{|D|}.$$

If D is empty, set $f_{10}(s) = f_{11}(s) = \frac{1}{2}$, again.

General Directions: These features describe whether the stroke is aligned from from top right to bottom left or from top left to bottom right. It takes the normalised vector form start to end point and projects it onto the two directions

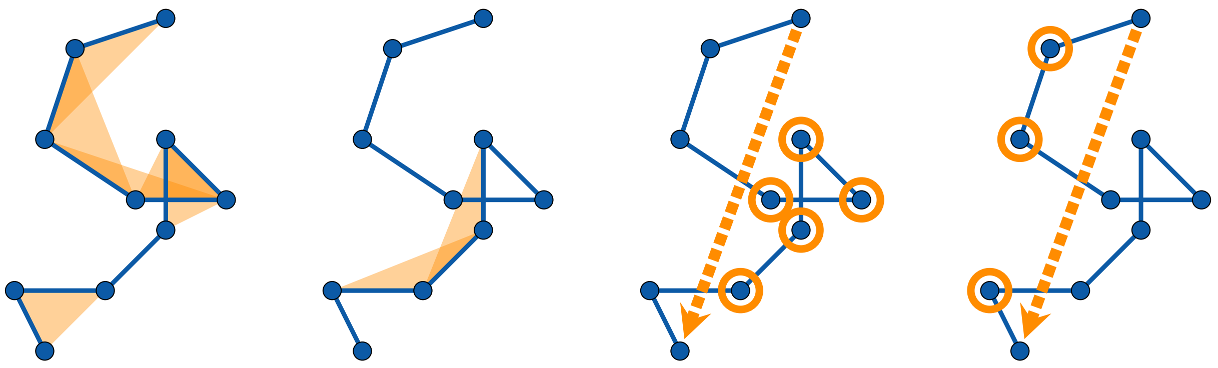


Figure 2.20: *Portrayal of:* LEFT *and* RIGHT CURVATURE, f_8 *and* f_9 *(the two images on the left), and* LEFT *and* RIGHT OF START-END LINE, f_{10} *and* f_{11} *(both two images on the right).*

we are interested in.

$$f_{12}(s) := \frac{(P_n - P_1) \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix}}{\sqrt{2} \cdot \|P_n - P_1\|_2},$$
$$f_{13}(s) := \frac{(P_n - P_1) \cdot \begin{pmatrix} -1 \\ 1 \end{pmatrix}}{\sqrt{2} \cdot \|P_n - P_1\|_2}.$$

We could store the vector $\frac{P_n - P_1}{\|P_n - P_1\|_2}$ directly, but we are interested in the given directions specifically. For a more general set-up to analyse such directional vectors, see Section 4.1.

START and END POINT: These are simply the coordinates of the start and end point.

$$f_{14}(s) := x_1,$$
$$f_{15}(s) := y_1,$$
$$f_{16}(s) := x_n,$$
$$f_{17}(s) := y_n.$$

START and END LOOP: These are two very heuristic features. They detect loops at the start and end of the stroke—i.e. if a point in the middle of the

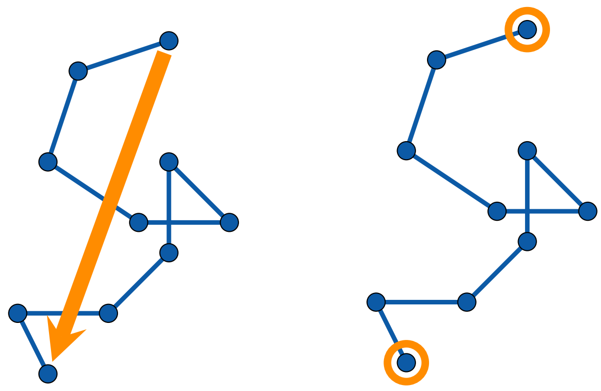


Figure 2.21: *Portrayal of:* General Direction $(f_{12}, f_{13})^T$ *(left),* Start Point (f_{14}, f_{15}) *and* End Point (f_{16}, f_{17}) *(right).*

stroke coincides with the start and end point, these features evaluate to 1. But in contrast to most other features, the values less than 1 do not have a concrete geometric interpretation. In particular, finding thresholds for these features to decide when these properties are identified as present in strokes can only be done by looking at samples. These two features are incredibly helpful to recognise the numbers `6` and `9`, but they also come in handy with `8` and `0` — especially when the ends of the stroke do not connect properly, as illustrated in Figure 2.22 below.

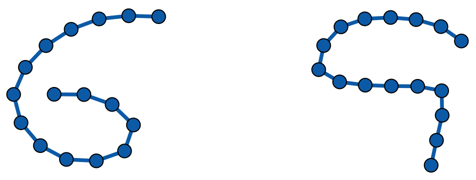


Figure 2.22: *Strokes that represent* `6` *and* `9`*, respectively, but whose ends do not properly form a loop.*

We compute these feature in the following way: Firstly, let $d_{i,j} = \left\|P_j - P_i\right\|_2$ be the Euclidean distance between the points P_i and P_j for any $i, j = 1, \ldots, n$. Secondly, set

$$a_s := \max\left\{j \in \{2, \ldots, n\} \mid d_{1,2} \le d_{1,3} \le \cdots \le d_{1,j}\right\}$$

and, analogously,

$$a_e := \min\left\{j \in \{1,\dots,n-1\} \mid d_{n,j} \geq \cdots \geq d_{n,n-2} \geq d_{n,n-1}\right\}.$$

They describe how far along the stroke one can walk away from the start (or end) point before turning back towards the start (or end). As an example, for the stroke in Figure 2.23 these values are $a_s = 5$ and $a_e = 6$. I.e., starting from P_1 we can walk up to P_5 before turning back towards P_1. Starting from P_{10}, however, we can walk up to P_6 without turning ever back towards P_{10}.

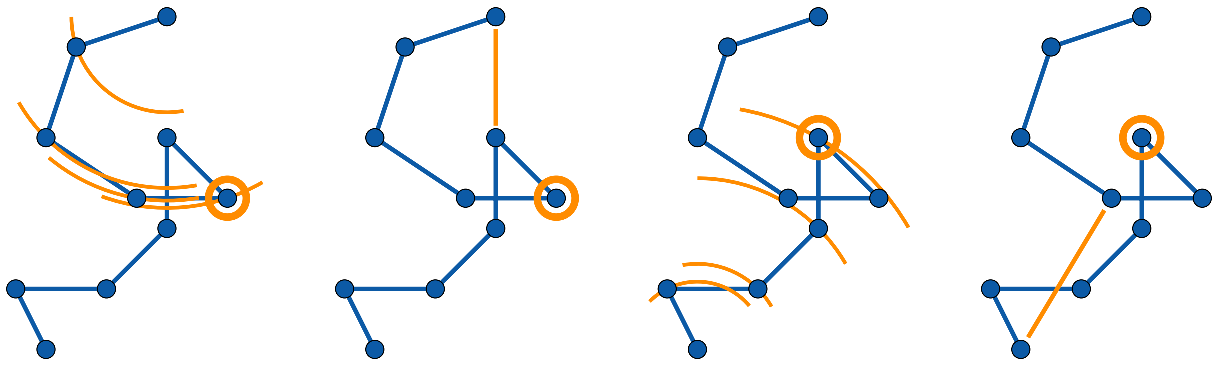


Figure 2.23: *Portrayal of: a_s and* Start Loop *f_{18} (the two images on the left), and a_e and* End Loop *f_{19} (the two images on the right).*

To detect a loop at the start and end of the stroke, we then check how close the stroke comes back to the start and end points; outside of the parts that initially move away from them. So, we get the features

$$f_{18}(s) := 1 - \frac{\min\left\{d_{1,j} \mid j > a_s\right\}}{\max\left\{d_{i,j} \mid i,j = 1,\dots,n\right\}},$$
$$f_{19}(s) := 1 - \frac{\min\left\{d_{n,j} \mid j < a_e\right\}}{\max\left\{d_{i,j} \mid i,j = 1,\dots,n\right\}}.$$

Center Traversal: This features describes whether the stroke runs through the center of the bounding box. Its primarily benefit is to distinguish `8` from `0`. They can, in theory, be held apart by Left and Right Curvature, but in practise those two features are often inconclusive.

$$f_{20}(s) := 1 - \frac{\min\{\|P_i\|_2 \mid i = 1, \ldots, n\}}{\sqrt{2}}.$$

We use half the length of the diagonal of the square $[-1, 1]^2$ as the normalisation factor instead of the maximum over all $\|P_i\|_2$, since it works better in practise.

Convex Hull ([11]): Considering the stroke as a set of n points, we can compute the convex hull of these points. Its area relative to the area of the bounding box (which is 4 here) is a measure of the complexity of the stroke. Let $\{Q_1, \ldots, Q_k\} \subseteq \{P_1, \ldots, P_n\}$ be the this convex hull.[16] In particular, these points Q_i are ordered counter-clockwise and $Q_k = Q_1$. With the well-known formula for the area of (convex) polygons given by a such specified order of its vertices, we get

$$f_{21}(s) := \frac{1}{4} \cdot \frac{1}{2} \cdot \sum_{i=1}^{k-1} \det(Q_i, Q_{i+1}).$$

Inflexions ([11]): Here we measure the signed distance from the midpoint between start and end point to the point in the middle of the stroke. This is a coarse measure for the symmetry and complexity of the stroke. As our strokes are (almost) equidistantly sampled, we simply use the index $\lfloor \frac{n}{2} \rceil$ for the point, that halves the length of the stroke. This leads to

$$f_{22}(s) := x_{\lfloor \frac{n}{2} \rceil} - \frac{1}{2} \cdot (x_1 + x_n),$$
$$f_{23}(s) := y_{\lfloor \frac{n}{2} \rceil} - \frac{1}{2} \cdot (y_1 + y_n).$$

Deviations ([11]): This computes the average distance of the points on the stroke to their center of mass, which we denote by $c = \frac{1}{n} \sum_{i=1}^{n} P_i$. It is equal to 1 if and only if the points lie on a circle.

$$f_{24}(s) := \frac{\frac{1}{n} \sum_{i=1}^{n} \|P_i - c\|_2}{\max\{\|P_i - c\|_2 \mid i = 1, \ldots, n\}}.$$

[16]There are many algorithm to compute the convex hull—we use *Graham scan*, see [17]

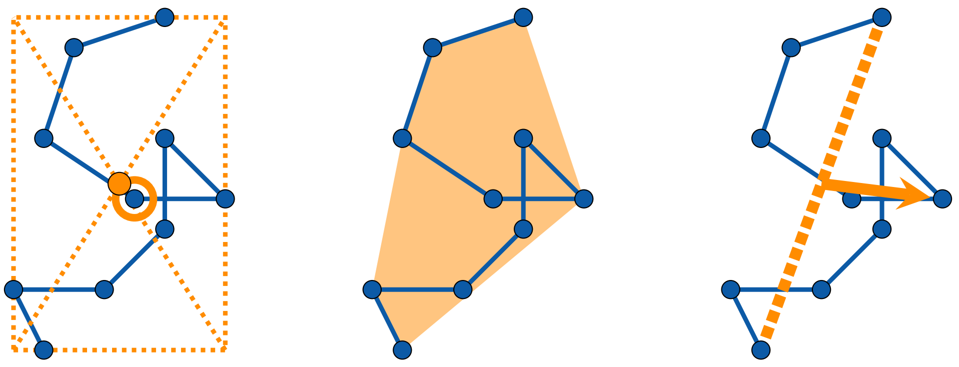


Figure 2.24: *Portrayal of:* CENTER TRAVERSAL f_{20} *(left),* CONVEX HULL f_{21} *(centre) and* INFLEXIONS (f_{22}, f_{23}) *(right).*

BULGES: The last feature counts the number of bulges on the right-hand side of the stroke. Basically, it is the number of local maxima of the x-coordinates. It was added to distinguish `3` from `7` and `1` and also to help differentiate `8` and `0`. We compute it in the following way. Set

$$B := \{i \in \{2, ..., n-1\} \mid x_{i-1} < x_i + 0.001 \quad \wedge \quad x_{i+1} < x_i + 0.001\}$$

and then

$$f_{25}(s) := \begin{cases} 1, & \text{if } |B| \geq 2, \\ 0, & \text{else.} \end{cases}$$

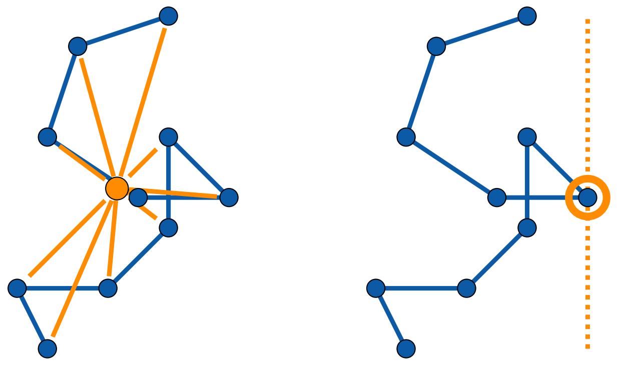


Figure 2.25: *Portrayal of:* DEVIATIONS f_{24} *(left) and* BULGES f_{25} *(right).*

These 25 features describe concrete geometric properties and most values have an explicit interpretation. For example, $f_{10}(s) = \frac{1}{2}$ means that half of the points on the stroke s are left of the start-end line and the other half is right of it — which is a characteristic of stroke types like 2. Despite the fact that many feature values can be re-translated into geometry, we want to focus on values close to 1 and 0 in the analysis. To emphasise these values we re-scale every feature by an S-shaped function. It is built by the modulation function μ_c introduced in Section 2.3.2. Its precise definition is given by

$$\rho_{c,\lambda} : [0,1] \to [0,1], \quad x \mapsto \begin{cases} \lambda \cdot \mu_{\frac{1}{c}}\left(\frac{x}{\lambda}\right), & \text{if } x \leq \lambda, \\ \lambda + (1-\lambda) \cdot \mu_c\left(\frac{x-\lambda}{1-\lambda}\right), & \text{if } x > \lambda. \end{cases}$$

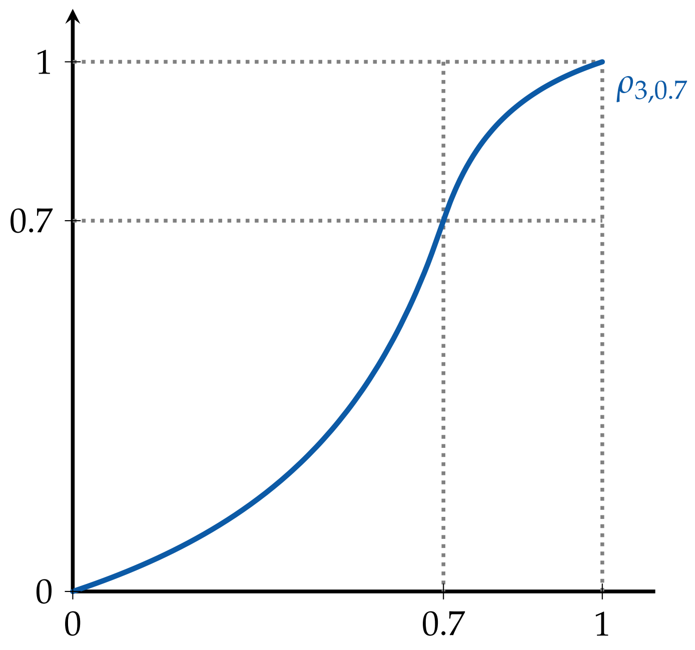


Figure 2.26: *Graph of* $\rho_{3,0.7}$.

In ALICE:HWR we use $\rho_{3,0.7}$. I.e., values larger than 0.7 are multiplied by 3 and value below 0.7 are divided by 3. The graph of this particular re-scaling function can be seen in Figure 2.26. The asymmetry of the break point $\lambda = 0.7$ allows for the use of features like f_{10} and f_{11} that are complementary by design and to measure slightly different things. E.g., $f_7 + f_6 \equiv 1$, which we can interpret as

a stroke is as much a straight line as it is not a closed loop and vice versa.

However, $\rho_{3,0.7} \circ f_6$ overestimates how straight an almost straight stroke is; and $\rho_{3,0.7} \circ f_7$ overestimates how closed an almost closed loop is.

The ideas and statements in the next three chapters will be independent from these concrete features. But we want to keep as intuitive examples in mind when we talk about geometric properties that might describe strokes.

3 Geometric transformations of strokes

As a tool, it defines what it can do, and, more importantly, what people will expect to do with it. If you buy a sports car, you're likely to get speeding tickets. If you have a hammer, everything looks like a nail.

— Steve Swink, *Game Feel*

One of the primary concerns of this thesis is to understand what strokes are by examining how they can be deformed. This allows us to generate new strokes in a way that preserves their overall form and meaning. We mainly use this to create more samples from small training sets — something motivated in Section 1.3 for ALICE:HWR. Moreover, this can be used in any form of handwriting synthesis to induce variance in computer-generated strokes.

This chapter deals with classes of transformations which are all, to some degree, motivated by practical considerations. We do not claim that these are all reasonable ways to deform strokes. However, as we will see in the end, they are sufficient to generate a wide variety of alterations and describe equivalence between them. Additionally, most of them preserve the shape of strokes as we will discuss in Chapter 5.

Beforehand, a point to pay attention to: When we talk about and show certain transformations, we will always apply them as they are. In practice, we get much better results when we re-sample the resulting strokes equidistantly. This can be tried out, however, in some of the widgets in the companion iBook [67].

3.1 Four classes of geometric transformations

One central characteristic of handwriting is that strokes which might look radically different can represent the same symbol. Take for example all the different ways to write the number 3 in Figure 1.6 below.

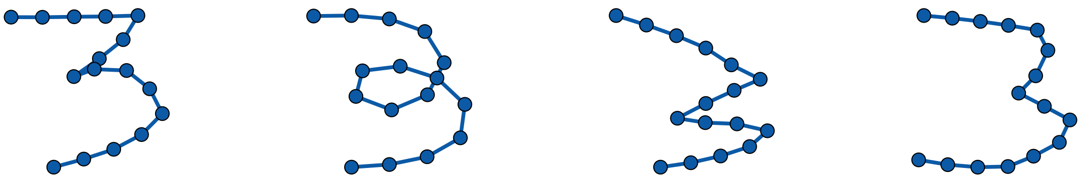


Figure 3.1: *Various different ways to write 3.*

Some notable differences are

— whether there is a loop or a tip in the middle,

— whether the line is built from circular arcs or straight lines and

— whether the upper end of the line points more up or down.

Recognising these individual parts is relatively easy. Also, as they are more or less independent from each other, they can be freely combined. Having both these points in mind, it is an obvious idea to build a database with all different kinds of 3's and to describe/recognise them by these different properties. It is, however, more interesting to find out whether there are universal underlying geometric properties which make a stroke a 3. E.g., the loop in the middle of the line is neither a sufficient nor a necessary condition for a stroke to be a 3. Apart from being an intriguing question in itself, this has a powerful, practical consequence: When a classification software — based for example on a neural network — is trained with hundreds or even thousands of strokes representing a 3 and when all of them have a loop in the middle, it is likely that this software cannot recognise a 3 written without a loop.[1] When the symbol 3 is radically

[1]Of course, this depends heavily on the classification method, the implementation of it and the underlying problem.

different from all other symbols the software shall classify, minor deviations in the strokes have a smaller impact on the performance. Nevertheless, it is a tremendous advantage if computer programs can automatically observe and respect this difference.

One can also consider the inverse problem: Given a sample set of strokes representing a 3, how can further examples for this symbol be generated which

1. look different than the existing samples, but
2. still look similar to them.

This allows us to synthesise handwriting nicely: we can give a computer a handwriting sample, and it produces text with the characteristics of this handwriting without the cookie-cutter aesthetics of normal computer fonts and typefaces.

These different thoughts lead to a mutual question: How can a stroke be deformed? In this chapter, we will look at various ways to do this and how it affects the shape of the strokes.

3.1.1 Projective transformations

When thinking about geometric transformations, the first things that come to mind are rotations, translations and reflections. The most basic class of these are Euclidean transformations. Building up from that, we get more and more complex transformations that change more and more geometric quantities like angles, aspect ratios, areas, and others. A very general class encompassing all these are projective transformations.[2]

In the context of HWR, they can be thought of as maps that change the orientation of strokes, but leave the shape intact. That is illustrated by Figure 3.2. This makes projective transformations essential objects to examine because every HWR algorithm should be able to handle at least small rotations, translations and shearings as they happen naturally while writing.

[2]The most general transformations are arbitrary bijections $\mathbb{R}^2 \to \mathbb{R}^2$, but they are not particularly useful when doing geometry.

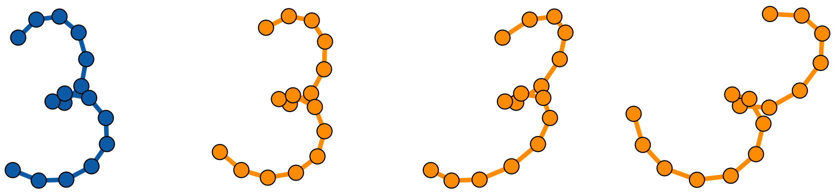


Figure 3.2: *Various transformations of a stroke: rotation, shearing and a general projective transformation.*

On a technical level, applying projective transformations to stroke is almost trivial: For any $M \in \mathrm{GL}_3(\mathbb{R})$ and $\hat{s} \in \hat{\mathfrak{S}}$ we set

$$M\hat{s} := \left(M\hat{P}_i\right)_{i=1}^{n}.$$

The only problem herein is that the image points $M\hat{P}_i$ have to be manually dehomogenised, since, in general, the last coordinate will not be equal to 1. But if we assume that this is done implicitly, this process is just a matrix-vector multiplication. What makes this interesting to consider is that we can interpret it in three slightly different ways.

The first was already mentioned in the last paragraph: Variations of stroke arise during writing and they can be often modelled as rotations and shearings. E.g., finding a suitable shearing parallel to the writing direction is an essential part of analysing longhand to straighten up the written letter. See, for example, [66]. In Chapter 5 we will define the **shape** of a stroke and see that projective transformations do not change it. In other words, this will mean that all elements in the orbit $\mathrm{GL}_3(\mathbb{R})s$ of any stroke s will be identified when considering their shape.

The second way projective transformations are interesting is simply the fact that they encompass all affine transformations of the drawing plane $\mathbb{R}^2$: Let $A \in \mathrm{GL}_2(\mathbb{R})$ and $b \in \mathbb{R}^2$. Then,

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto A \begin{pmatrix} x \\ y \end{pmatrix} + b$$

is an affine transformation of $\mathbb{R}^2$. And on the finite points of $\mathbb{RP}^2$ in standard embedding we get the same map via

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} A & b \\ \mathbf{0}^T & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Reversely, every projective transformation that is, up to a scalar multiple, of the form

$$\begin{pmatrix} * & * & * \\ * & * & * \\ 0 & 0 & 1 \end{pmatrix}$$

represents an affine transformation. But that means that all maps we use to normalise strokes — translations, rotations, dilations — can be written as simple matrix multiplications. That makes it easier to implement them and, subsequently, to work with centred and nailed strokes.

The last point that makes projective transformations interesting is that they can be used the create new samples for stroke types. In Example 3.2.2 we do this with other transformations, too. These other ones are explicitly motivated by what strokes are qualitatively. Projective transformation, however, are applied to individual points. In particular, any point on a stroke might get mapped to the line at infinity. And even if that is not the case, issues might emerge when rendering projectively transformed strokes on screen, since this is done via the line segments between consecutive points on the strokes.

It is a bit tricky to define line segments in projective space. However, we can use the cross-ratio of four collinear points to do so. The cross-ratio is one of the most fundamental and ubiquitous tools from projective geometry. It describes the ratio of ratios of lengths and is the simplest projectively invariant function. Among other things, it allows to continuously parametrise any line by $\mathbb{R} \cup \{\infty\}$ in a canonical way. Using it, a line segment can be represented as an interval in this parametrisation.[3]

Now, consider the four (finite) points A, B, C, D and the very simple stroke they represent at the top of Figure 3.3.

[3]Please see [56] and [55] for more on that.

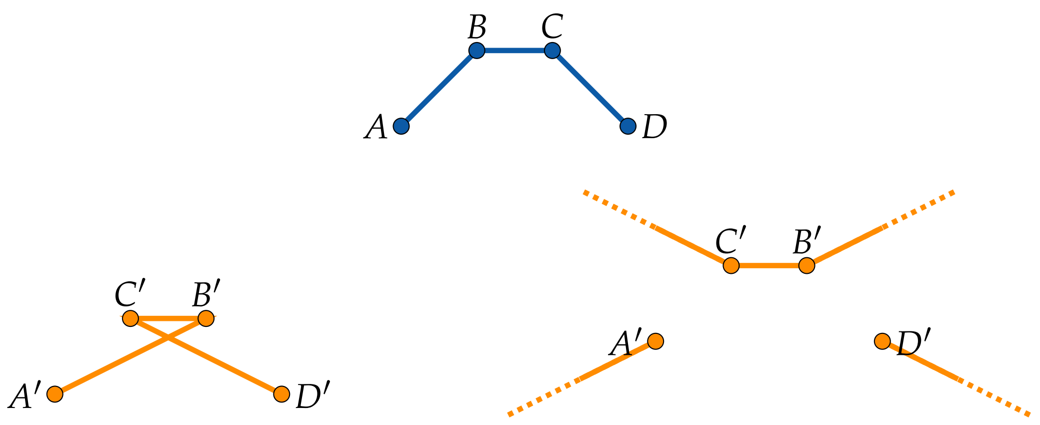


Figure 3.3: *A transformed stroke with ambiguous rendering.*

As no three of them lie on a line, we can find a projective transformation that fixes A and D and interchanges B and C. It is, however, not clear how a computer should render the associated stroke on a screen. We could decide that every image point A', B', C', D' gets connected to its successor (or predecessor) via the direct straight line between them. This is the bottom left version in the figure. We could also map every point on the line segments of the original stroke A, B, C, D via the projective transformation; leading to the bottom right version in which some line segments pass through infinity.[4]

Consequently, when we use projective transformation as a graphics tool — to generate strokes that "look" the same — we demand that no line segment of the stroke should intersect the line at infinity after a projective transformation is applied. We do not know where a line segment of an arbitrary stroke might lie though. Thus, we demand two things when distorting strokes graphically:

— We apply general projective transformations only to centred strokes.

— We want them to fix the origin.

— No point in the interior of the normalised bounding box spanned by $\left(-\frac{1}{2}, -\frac{1}{2}\right)$ and $\left(\frac{1}{2}, \frac{1}{2}\right)$ should get mapped to the line at infinity.

The first two conditions are no real constraint, since normalising into the set of centred stroke $\mathfrak{Q}$ and back out of it is achieved by conjugation with a

[4]From a purely geometric point of view, without the context of handwritten/-drawn symbols or any other practical application, this second version is the canonical one for mapping line segments.

translation-cum-dilation; and this is a projective transformation again. This means that we can always assume that we have a representative for such a transformation of the form

$$M = \begin{pmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{pmatrix}.$$

The third condition is equivalent to saying that the pre-image of the line at infinity should not intersect the bounding box. As the transformation M operates on lines via $l \mapsto \left(M^{-1}\right)^T l$, the pre-image of the line at infinity is

$$M^T l_\infty = M^T \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} c \\ f \\ 1 \end{pmatrix}.$$

As a line in $\mathbb{R}^2$, it has the normal vector (c, f). And a point (x, y) lies in the half space bounded by the line and into which the normal vector points if $cx + fy + 1 < 0$. We can insert the coordinates of the four corners of the normalised bounding box to get the four inequalities

$$\begin{aligned} \frac{1}{2}c + \frac{1}{2}f + 1 &> 0, \\ -\frac{1}{2}c + \frac{1}{2}f + 1 &> 0, \\ \frac{1}{2}c - \frac{1}{2}f + 1 &> 0 \text{ and} \\ -\frac{1}{2}c - \frac{1}{2}f + 1 &> 0, \end{aligned}$$

because these vertices should *not* lie in those half spaces. After multiplying with -2 they can be summarised to $|c| + |f| < 2$. And because the bounding box is convex, it is enough to find conditions for the corners.

Definition 3.1.1: A projective transformation M is **compliant** if it is of the form

$$M = \begin{pmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{pmatrix},$$

it fulfils

$$|c| + |f| < 2.$$

△

To summarise: When we use projective transformations as a mathematical tool for normalisation, we can simply apply them pointwise to strokes. But using them to distort strokes seen as images, we have to restrict ourselves to compliant projective transformations.

3.1.2 Convex combinations

The basic stroke space we use, $\mathfrak{S} = \left(\mathbb{R}^2\right)^n$, is an affine space. In particular, we can compute affine combinations of strokes and get a new stroke as a result. We could consider arbitrary linear combinations of strokes, too, but in practise this is less descriptive: Let $s_1, ..., s_m$ be strokes, $\lambda_1, ..., \lambda_m$ be real coefficients and set $r := \sum_{i=1}^{m} \lambda_i s_i$. After we set $\Lambda = \sum_{i=1}^{m} \lambda_i$ we can rewrite r as

$$r = \Lambda \cdot \sum_{i=1}^{m} \frac{\lambda_i}{\Lambda} s_i.$$

That means that r emerges as an affine combination of the s_i which then get scaled by the factor L. This scaling has its centre at the origin of the drawing plane $\mathbb{R}^2$. When we now translate every point of every stroke s_i by the vector $t = \in \mathbb{R}^2$, the linear combination r gets translated by Lt. So, arbitrary linear combinations of strokes are not nonsensical per se, but a little bit impractical.

When we confine ourselves to affine combinations of strokes, there is the particularly interesting subclass of convex combinations. When two strokes represent the same symbol, and their writing styles are close enough, then every convex combination of them will also represent this symbol. This is illustrated in Figure 3.4.

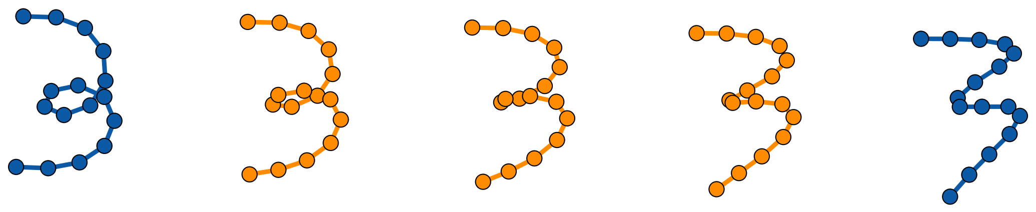


Figure 3.4: *Several convex combinations "between" two different strokes representing a 3.*

It is nigh impossible to formalise this in a way that is useful for theoretical considerations. The main reason is that the semantics of a stroke are ambiguous. For example, in Figure 3.4 we can simply define that the far-left and far-right strokes represent something different.

With the sample dispersals we introduced in Definition 2.2.3 we could introduce the following additional property: A sample dispersal $\mathcal{D}$ on $\mathfrak{S}$ is **high-concave** if for all fuzzy sets $m \in \mathcal{D}$ there exists a $\lambda_m \leq 1$ such that all the super-level sets

$$\{s \in \mathfrak{S} \mid m(s) \geq \lambda_m\}$$

are convex.

This would mean that convex combinations of "good" representations of a certain symbol are again "good". We will not use this in any strict sense though—but we want to keep it in mind as a general model assumption.

Note that this heavily depends on the aforementioned 'style' in which the strokes are written. In Figure 3.5 we see two ways of writing a 3 for which convex combinations do not necessarily result in new 3's. So, if both initial strokes are part of the training data for an HWR algorithm, they should probably be associated with different stroke types.

When we consider affine combinations that are explicitly not convex we can adapt our interpretation. Consider two strokes s, t and an affine combination $r = \lambda s + (1 - \lambda)t$. The closer λ is to 1 the more the stroke r will look like s. When λ approaches 1 from below, every "geometric property" that makes a stroke look like t will vanish, and every property present in s will be brought

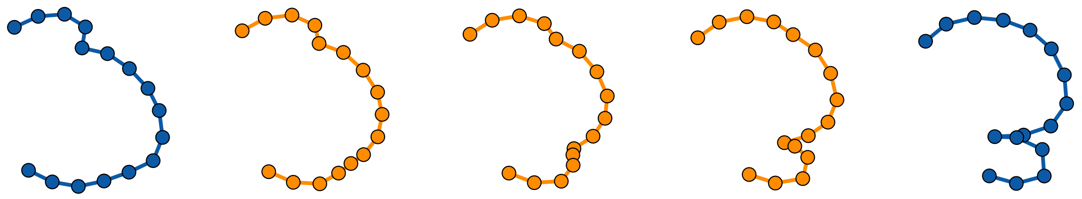


Figure 3.5: *Convex combinations "between" two representative of* 3*, which are incompatible. I.e., they form non-*3 *strokes.*

out. When λ now is larger than 1 we can think of these s-properties being exaggerated relative to t. This is illustrated in Figure 3.6 with t representing an S and s representing a 3. The bottom half of these strokes are, more or less, the same while the upper half gets mirrored in-between and blown-up once λ is larger than 1. (Increasing λ is depicted by going to the right in the figure.)

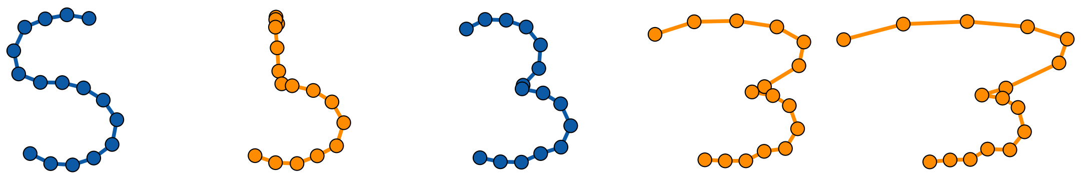


Figure 3.6: *Affine combinations exaggerating a* 3 *(centre) relative to an* S *(left).*

We can look at this in a slightly different way: Let s be any stroke and let u be the straight line segment between the start and end point of s such that all its points are uniformly distributed along this segment. We will see this object much more in the next section. For now, we do interpret it as being featureless: In Example 2.2.6 we have seen that the Straightness of a stroke is a feature that detects such line segments. As we try to work with equidistantly sampled strokes in praxis, a Straightness value of 1 uniquely characterises such strokes u. Moreover, with the exact positions of points on u known, we can compute any other feature we might be interested in. So, a single specific value of a single specific feature determines all other feature values. In this sense, straight line segments are featureless: other features have no degree of freedom left.[5]

[5]Some features might still be sensitive to different point distributions along the straight line segment. But recall that we usually work with (almost) equidistantly sampled strokes. So, in practice, other feature values are most often fixed once Straightness$(s) = 1$.

If we now consider affine combinations $r = \lambda s + (1-\lambda)u$, we can imagine that the geometric properties of s diminish the closer λ gets to 0 and that they get exaggerated when λ is larger than 1. In Figure 3.7 this is shown for a stroke representing a 3.

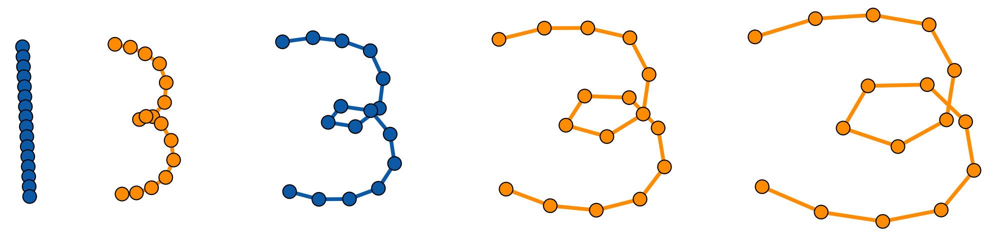


Figure 3.7: *Affine combinations exaggerating a 3 (centre) relative to a "featureless" stroke (left).*

When we see these affine combinations as functions in s, they form a group. To be precise:

Lemma 3.1.2: *For any stroke $s \in \mathfrak{S}$ let u be the straight line segment between P_1 and P_n given by a uniform point distribution. For any $\lambda \in \mathbb{R}$ define the* ***affine pull*** *function*

$$C_\lambda : \mathfrak{S} \to \mathfrak{S}, \quad s \mapsto \lambda s + (1-\lambda)u.$$

Then, $\{C_\lambda \mid \lambda \in \mathbb{R}^\times\}$ forms a group under concatenation with $C_\lambda \circ C_\mu = C_{\lambda\mu}$.

The computations to show this are straight foreward and so we skip them.

There are many more ways in which convex and affine combinations of strokes make sense; one of which we will see in the next section. But even without any further applications, it is clear that the underlying affine structure of our basic stroke space $\mathfrak{S}$ will be useful.

As a final note on convex combinations: when we see them as a graphical/artistic way to morph between strokes, there are numerous ways to do this. One that is popular in animations to transition between keyframes uses cubic splines; in particular, Catmull-Rom splines as we introduced them in Section

2.3.2. However, convex combinations provide a more direct way, literally and figuratively, that is more useful for data analysis.

3.1.3 Accelerations

Another way to transform strokes comes from actually writing them by hand: we can ask the question what happens if a stroke gets written faster. The main problem therein is to define what "faster" means and in extension what the "writing speed" is. The geometric effect we want to emulate is the one shown in Figure 3.8.

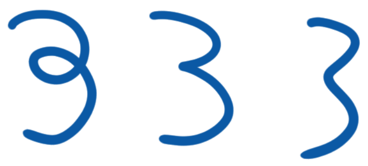


Figure 3.8: *Writing a stroke "faster".*

When a 3 is drawn with a loop in the middle and we write it "faster", we can expect the loop turn into a cusp and then into a smooth bulge. We model this process for general strokes via the following heuristic rule:

When writing a stroke, the user tries to get from start to end point as fast as possible. Any deviation from the straight start-end line must be an essential part of the stroke.

It directly suggests establishing a local coordinate system for every stroke in which one direction points from start to end point and the other one is perpendicular to it. In other words, we normalise each stroke via a similarity map such that the start point lies at the origin and, for convenience, that the endpoint lies at $(1,0)$. In Section 2.2.1 we introduced the notion of nailed strokes. With it we can write the set of all strokes with the start and end point as just described by $\mathfrak{N} := \mathfrak{N}_{(0,0),(1,0)}$. In particular, the "writing-faster" process will only affect the x-coordinates in this set-up.

In order to understand this faster writing, we will assume that the points on a stroke are produced uniformly in time: point P_i is "written" after $i-1$ time units. That means that the difference between x-coordinates of consecutive can be interpreted as the speed between them in the start-end direction. Writing faster in start-end direction then would mean to add a constant $\Delta v \in \mathbb{R}$ to each of them.[6] So, we want to replace

$$x_{i+1} - x_i$$

by

$$(x_{i+1} - x_i) + \Delta v.$$

To achieve this, we map

$$x_i \mapsto x_i + (i-1) \cdot \Delta v$$

for all i. But to get back a stroke from $\mathfrak{N}$ again we have to compress it in x-direction. The new end point has position $1 + (n-1) \cdot \Delta v$ and hence we have to divide each x-coordinate by $1 + (n-1) \cdot \Delta v$.

In total, we get a map $A_{\Delta v} : \mathfrak{N} \to \mathfrak{N}$ that depends on the parameter $\Delta v \in \mathbb{R}$ and which maps

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \longmapsto \frac{1}{1+(n-1)\cdot\Delta v} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} + \frac{\Delta v}{1+(n-1)\cdot\Delta v} \begin{pmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ n-2 \\ n-1 \end{pmatrix}$$

and which leaves the y-coordinates unchanged.

Definition 3.1.3: We call a function on $\mathfrak{N}$ of the above form an **acceleration** by Δv. $\triangle$

[6]We get the interpretation we want with $\Delta v > 0$, but we can formulate everything with negative Δv, too. The interpretation then changes to "writing slower".

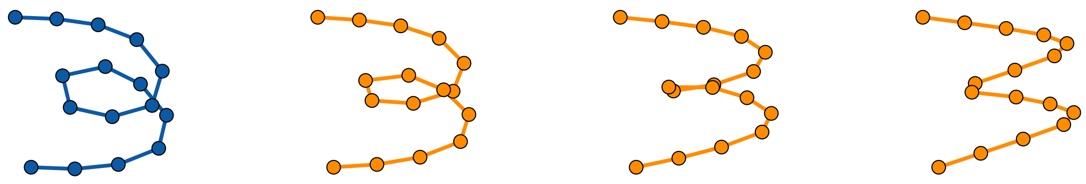


Figure 3.9: *Accelerating a stroke. Note that the stroke has to be normalised to $\mathfrak{N}$ for the above definition to work. In other words, the x-axis in this figure runs from start through end point of the stroke.*

Next, we look at is how iterated accelerations change strokes.

Theorem 3.1.4: *Let $A : \mathfrak{N} \to \mathfrak{N}$ be an acceleration by $\Delta v > 0$ and let $s \in \mathfrak{N}$.*

(1.) The sequence $\left(A^k(s)\right)_{k=1}^{\infty}$ converges towards a stroke $A^\infty(s)$ whose x-coordinates are uniformly distributed along the unit interval $[0, 1]$. I.e., the i-th x-coordinate of $A^\infty(s)$ is $\frac{i-1}{n-1}$.

(2.) The limit stroke $A^\infty(s)$ has no loops.

Proof. (1.) When we look at the second summand in the definition of accelerations, we can rewrite it as

$$\frac{\Delta v}{1 + (n-1) \cdot \Delta v} \begin{pmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ n-2 \\ n-1 \end{pmatrix} = \frac{(n-1) \cdot \Delta v}{1 + (n-1) \cdot \Delta v} \begin{pmatrix} \frac{0}{n-1} \\ \frac{1}{n-1} \\ \frac{1}{n-1} \\ \vdots \\ \frac{n-2}{n-1} \\ \frac{n-1}{n-1} \end{pmatrix}.$$

The vector on the right gives the uniform distribution of x-coordinates. Moreover, we see that an acceleration is just a convex combination on x-coordinates of the original stroke and this uniform distribution. As the latter one is fixed and has the same start and end point as the stroke we apply the acceleration to, we can write this map as an affine pull[7] C_λ with

[7]This is not entirely correct, because accelerations only operate on x-coordinates. But since affine pulls operate on x- and y-coordinates separately, we can disregard this and use the group structure of affine pulls for accelerations.

$\lambda = \frac{1}{1+(n-1)\Delta v} < 1$ as defined in Lemma 3.1.2. In particular,

$$C^k_\lambda(s) = C_{\lambda^k}(s) \to C_0(s) \quad \text{for } k \to \infty$$

and $C_0(s)$ is the uniform distribution of x-coordinates.

(2.) For a loop to form in a stroke we need to have indices $i < j$ with $x_i > x_j$. But this is impossible for $A^\infty(s)$, due to part (1.). $\square$

Looking at Figure 3.9 and Theorem 3.1.4 we see that accelerations do embody the desired effect shown in Figure 3.8. In particular, they eliminate any loop if applied often enough. For a 3 this is desirable, but for other strokes like 6 or φ this becomes problematic as their loops are essential to their form and their recognisability.

Even worse, strokes which are closed curves — i.e. when start and end point are very close to each other — get deformed very quickly. Both these problems with essential loops can be observed in Figure 3.10 below.

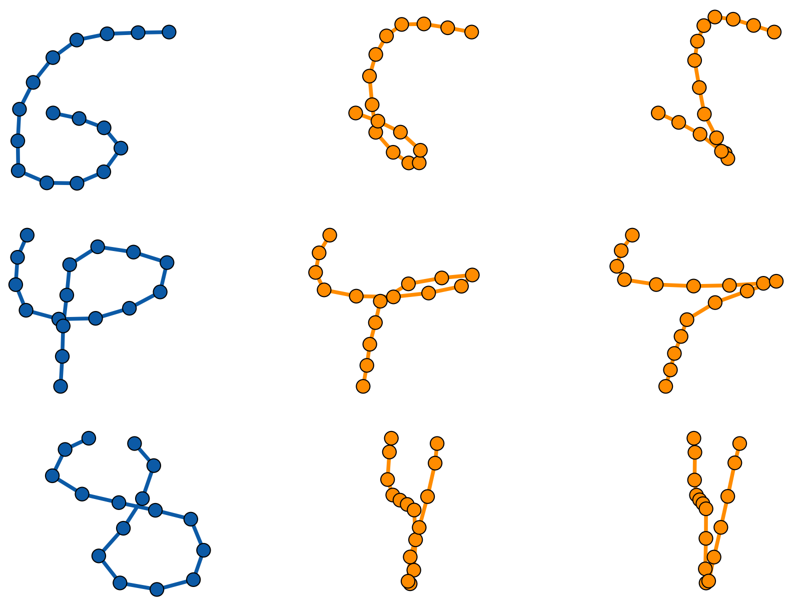


Figure 3.10: *Accelerating strokes with essential loops.*

In the proof of Theorem 3.1.4 we saw that accelerations are just special convex combinations of strokes that operate only on x-coordinates. In particular, they inherit the group structure from affine pull functions. But as we want to apply accelerations in the form defined here, we state their multiplication and inversion formulae explicitly to understand how the parameter Δv effects this operations.

Lemma 3.1.5: *The set $\{A_{\Delta v} \mid \Delta v \in \mathbb{R}\}$ forms a group under concatenation. In particular, $A_0 = \mathrm{id}_{\mathfrak{S}}$ is the neutral element,*

$$A_{\Delta v} \circ A_{\Delta w} = A_{(n-1)(\Delta v + \Delta w + (n-1)\Delta v \Delta w)} \quad \textit{and}$$
$$A_{\Delta v}^{-1} = A_{-\frac{\Delta v}{1+(n-1)\Delta v}}.$$

3.1.4 Gaussian filters

Gaussian filters are a common tool in Signal Processing and are used to reduce noise and details in signals. In the realm of computer graphics they are often known as Gaussian blur, and this is where we draw our motivation from. Seeing a picture as a function that maps every point to a grey value — represented for example by numbers in the interval $[0, 1]$ — a Gaussian filter replaces this function by a convolution with a kernel that, in the one-dimensional case, looks like this:

$$x \mapsto \sqrt{\frac{a}{\pi}} \cdot e^{-ax^2}$$

To represent pictures on a computer, they are usually modelled by pixels: Small squares in a grid coloured in a single grey value. In this discretisation, the integral in the convolution becomes a sum and the weights obtained from the Gaussian kernel have to adapt to this finite case. See for example [59] for this graphical application.

What we take away from this is the rule

Replace every point by a weighted sum of the surrounding points.

And we will apply this, of course, to strokes. We will use this in the two different ways already mentioned above: To reduce noise and to reduce detail.

A Gaussian filter, applied once or twice, leaves a stroke unchanged except

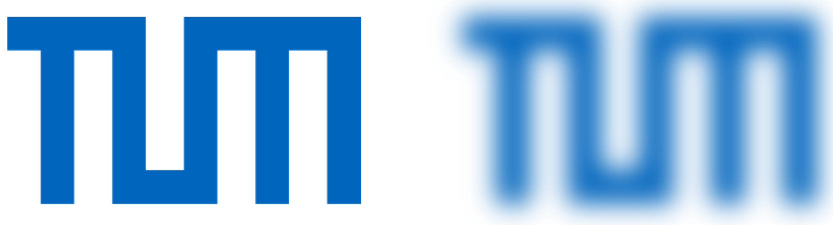


Figure 3.11: *Gaussian blur applied to the TUM logo in the program* Krita.

for cleaning it from little errors. They often come from the "trembling" hand of the user, especially when writing/drawing slowly, and the imperfect recording process. The resulting stroke is much smoother, and the computation of features is more precise and stable. Applied over and over again a Gaussian filter will eliminate all distinct properties. For computer graphics, this means that in the limit the image will be filled by a single grey values. Our strokes will turn into a straight line segment as we will see.

The reason why this is relevant for handwriting recognition is that after applying a Gaussian filter often but not too often a stroke will have lost all irrelevant geometric properties but still will have retained all significant ones. So, one can say that Gaussian filters (with some minor assumptions) alter the local shape of strokes, but preserve the global shape and orientation. In this sense, they complement projective transformations which preserve the spatial relations between the individual points but alter the overall orientation. We will see this interplay between Gaussian filters and projective transformations in Examples 3.2.2 and 3.2.3.

The next pages will define Gaussian filters and how their iterated application changes strokes. To make computation a little bit easier we will restrict our considerations to a subset of our stroke space $\mathfrak{S} = \left(\mathbb{R}^2\right)^n$.

Fix two distinct points $S, E \in \mathbb{R}^2$. Recall that we call a stroke $(P_i)_{i=1}^n \in \mathfrak{S}$ nailed if $P_1 = S$ and $P_n = E$ and that the set of nailed strokes is denoted by $\mathfrak{N}_{S,E} = \mathfrak{N}$. Later, we will also need the following: We define the set $\mathfrak{N}'$ of **truncated** nailed strokes by forgetting the first and last point of each stroke, as they are always S and E. So, as a set, it is given by $\left(\mathbb{R}^2\right)^{2(n-2)}$. To emphasise the roles in truncated strokes, we index their points from 2 to $n-1$ instead of from 1 to $n-2$.

In the rest of this section we define Gaussian filters as maps on $\mathfrak{S}$, but for the analysis of these, we will confine ourselves to $\mathfrak{N}$. In the end, this will be resolved in Theorem 3.1.18 in which we will let S and E vary. But for now, consider these two points explicitly given.

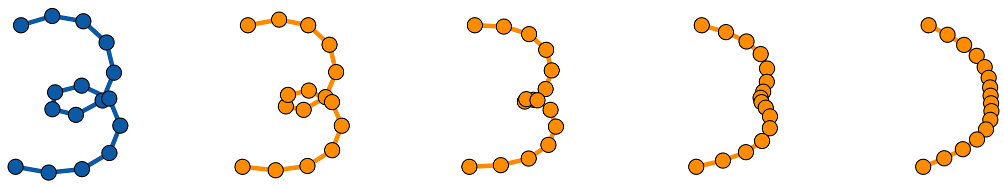


Figure 3.12: *A stroke under iterated application of a Gaussian filter.*

The most general definition of a Gaussian filter is the one given below. However, we will soon assume additional properties in order to have both a useful geometric interpretation and enough structure to deduce results.

Definition 3.1.6: A **Gaussian filter** is a map

$$\mathcal{G} : \mathfrak{S} \to \mathfrak{S}, \qquad s = (P_i)_{i=1}^{n} \quad \mapsto \quad \mathcal{G}s = \left(\sum_{j=1}^{n} a_{ij} P_j \right)_{i=1}^{n}$$

with real-valued coefficients a_{ij} such that

$$\sum_{j=1}^{n} a_{ij} = 1 \quad \text{for all} \quad i = 1, ..., n.$$

Writing the coefficients a_{ij} into a matrix $A \in \mathbb{R}^{n \times n}$, we will write $\mathcal{G}_A$ for the associated Gaussian filter and call A the **coefficient matrix** of the Gaussian filter. By abuse of notation we write $\mathcal{G}P_i$ for the i-th point of $\mathcal{G}s$; i.e., for $\sum_{j=1}^{n} a_{ij} P_j$. $\triangle$

A Gaussian filter replaces every point by an affine combination of its neighbours. Non-affine linear combinations would lead to the points on the stroke to drift towards infinity.

When we write strokes s by separating their x- and y-coordinates into s_{sep}, we observe that $\mathcal{G}_A(s_{sep}) = \left(\begin{smallmatrix} A & 0 \\ 0 & A \end{smallmatrix}\right) s_{sep}$. So, on *whole* strokes Gaussian filters are just linear functions. The crucial additional properties of Gaussian filters are the following.

Definition 3.1.7: We call a Gaussian filter $\mathcal{G}_A$ **nailed** if it fixes start and end point of a stroke. I.e. when $a_{1j} = \delta_{1j}$ and $a_{nj} = \delta_{nj}$ for all $j = 1, ..., n$. We call a Gaussian filter $\mathcal{G}_A$ **strict** if all coefficients a_{ij} are in the interval $[0, 1]$. △

First, we want Gaussian filters to be nailed as otherwise the start and end point might converge to the same point.

Second, using arbitrary affine combinations might blow up the stroke, and this contradicts the interpretation of Gaussian filters as smoothing functions. The strictness is the most obvious assumption to get a contraction-like behaviour.

Definition 3.1.8: Let $\mathcal{G}_A$ be a Gaussian filter. We call it **elastic**[8] if for all $i = 2, ..., n-1$ there exist $1 \leq j_- < i < j_+ \leq n$ with both $a_{j_-} \neq 0$ and $a_{j_+} \neq 0$. △

That means that every point P_i of a stroke s (that is not the start or end) gets replaced by a linear combination in which points both to the left and to the right of P_i occur. Iterating an elastic and strict Gaussian filter leads to a contraction towards the start and end points of the stroke as if rubber bands connected adjacent points. (Therefore the name 'elastic'.) This leads to the converging behaviour we want.

In practice, this is a natural assumption, because a Gaussian filter shall smooth/simplify a stroke by averaging over the points. And this only makes sense when points both to the left and to the right are accounted for.

[8]This term here has nothing to do with elastic curves or any other concepts with the same name.

Example 3.1.9: The most simple example of an elastic, nailed and strict Gaussian filter is the one which only takes direct left and right neighbours into account and assigning them the same weight. It is given by the matrix

$$A_\alpha := \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \alpha & 1-2\alpha & \alpha & 0 & \cdots & 0 \\ 0 & \alpha & 1-2\alpha & \alpha & \cdots & 0 \\ & \ddots & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & \alpha & 1-2\alpha & \alpha \\ 0 & \cdots & 0 & 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{n \times n}$$

with $\alpha \in \left(0; \frac{1}{2}\right]$ and we call Gaussian filters of this form **minimal**. Please think of them as the main representative of elastic, nailed and strict Gaussian filters. Moreover, these are the ones used throughout ALICE:HWR.

Note that we want $\alpha \neq 0$, since $A_0 = I_n$ and $\mathcal{G}_{A_0} = \mathrm{id}_{\mathfrak{S}}$. △

Up until now, all assumptions regarding Gaussian filters were natural and motivated by applications in handwriting recognition: They should be nailed, in order to keep the strokes normalised to the same start and end point and to keep them comparable. The strictness and elasticity both guarantee that the stroke contracts under the Gaussian filter.

Proposition 3.1.10: *Let $\mathcal{G}_A$ be an elastic, nailed and strict Gaussian filter. Then any fixed stroke of $\mathcal{G}_A$ is a straight line segment — i.e. a stroke in which every point lies on line spanned by start and end point.*

Proof. Consider a stroke, that is pointwise fixed by $\mathcal{G}_A$. I.e., for all $i = 2, ..., n-1$ we have

$$\mathcal{G}_A P_i = \sum_{j=1}^{n} a_{ij} P_j = P_i.$$

The right-hand equality can be re-written to get

$$P_i = \frac{1}{1 - a_{ii}} \sum_{\substack{j=1,\ldots,n \\ j \neq i}} a_{ij} P_j. \tag{3.1}$$

Note that $a_{ii} \neq 1$ by the assumption of $\mathcal{G}_A$ being elastic. As $\sum_{j=1}^{n} a_{ij} = 1$, the sum

$$\sum_{\substack{j=1,\dots,n \\ j \neq i}} a_{ij}$$

equals $1 - a_{ii}$. Thus, Equation (3.1) for P_i means that this point lies in the convex hull of all other points of the stroke. And as i was arbitrary, this is true for every single point. From this we can easily deduce, that all points must lie on the line from P_1 to P_n: Assume that the convex hull of all points on the stroke as more than two vertices. If we look at one particular vertex P_i of this convex set, the equation above tells us that it can be written as a convex combinations of other points in the convex set. But then it cannot be a vertex by definition. □

Remark 3.1.11: To put the last proposition into perspective, we will look at the following inversion: Given a line segment s, i.e. a stroke s in which every point P_i lies between P_1 and P_n. Then there exists an elastic, nailed and strict Gaussian filter $\mathcal{G}_{A_s}$ with s as its fixed stroke and we can write it down it explicitly: For points P_i on a line segment we can write $P_i = \lambda_i P_1 + (1 - \lambda_i) P_n$ for a suitable $\lambda_i \in (0, 1)$. Setting

$$A := \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ \lambda_2 & 0 & \cdots & 0 & 1 - \lambda_2 \\ \vdots & \vdots & & \vdots & \vdots \\ \lambda_{n-1} & 0 & \cdots & 0 & 1 - \lambda_{n-1} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix},$$

it is easy to check the the associated Gaussian filter $\mathcal{G}_A$ is elastic, nailed and strict and has the fixed stroke s. △

Remark 3.1.12: Another interesting question is how the fixed strokes of an elastic, nailed and strict Gaussian filter $\mathcal{G}_A$ look like. After all, there are many different ways points can lie on a line and Proposition 3.1.10 does not tell whether they even exists.

When all points of a stroke lie on a line, we can write every point P_i as $P_i = \lambda_i P_1 + (1 - \lambda_i) P_n$, for a $\lambda \in \mathbb{R}$. Then the condition for every point being fixed by $\mathcal{G}_A$,

$$\sum_{i=1}^{n} a_{ij} P_j = P_i \qquad \forall i,$$

can be re-written as

$$\left(\sum_{i=1}^{n} a_{ij}\lambda_j\right) \cdot P_1 + \left(\sum_{i=1}^{n} a_{ij}(1-\lambda_j)\right) \cdot P_n = \lambda_i P_1 + (1-\lambda_i)P_n \qquad \forall i.$$

Now, when we set $\Lambda := (\lambda_1, ..., \lambda_n)^T$, the last equation can be rewritten as

$$A\Lambda = \Lambda \qquad \text{and} \qquad A(\mathbf{1}-\Lambda) = (\mathbf{1}-\Lambda). \tag{3.2}$$

By the definition of Gaussian filters, A is a (right-)stochastic matrix, i.e. $A\mathbf{1} = \mathbf{1}$, and thus the second equation in (3.2) is actually identical to the first. And this of course means that the fixed strokes of $\mathcal{G}_A$ correspond to the eigenvectors of A to the eigenvalues 1. △

Before we analyse how Gaussian filters operate on arbitrary strokes, we give an example to illustrate the last three statements.

Example 3.1.13: Consider a minimal Gaussian filter $\mathcal{G}_{A_\alpha}$ with

$$A_\alpha = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \alpha & 1-2\alpha & \alpha & 0 & \cdots & 0 \\ 0 & \alpha & 1-2\alpha & \alpha & \cdots & 0 \\ & \ddots & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & \alpha & 1-2\alpha & \alpha \\ 0 & \cdots & 0 & 0 & 0 & 1 \end{pmatrix}.$$

As stated in Example 3.1.9, this Gaussian filter is elastic, nailed and strict. We can define a vector

$$\Lambda = (0, 1, ..., n-1)^T.$$

Independent of α, the i-th entry of $A_\alpha\Lambda$, for $i = 2, ..., n-1$, is

$$(A_\alpha\Lambda)_i \;=\; \alpha(i-2) + (1-2\alpha)(i-1) + \alpha i \;=\; i-1.$$

Accordingly, $A_\alpha \Lambda = \Lambda$ and every multiple of Λ represents a fixed stroke. If we scale it by $\frac{1}{n-1}$ we get a stroke with

$$P_i = \frac{i-1}{n-1} \cdot P_1 + \left(1 - \frac{i-1}{n-1}\right) \cdot P_n.$$

It is now an easy calculation to see that the distance between subsequent points P_i and P_{i+1} is exactly $\frac{\|P_1 - P_n\|_2}{n-1}$ and this implies that the uniforms distribution of points along the line segment from P_1 to P_n is a fixed stroke. △

To analyse how Gaussian filters change strokes we use the representation introduced in Section 2.2.1 in which we list first all x-coordinates of the points on the stroke and then all y-coordinates.

Definition 3.1.14: Given a matrix $A = (a_{ij})_{i,j=1}^{n} \in \mathbb{R}^{n \times n}$, we define the matrix A' by

$$A' := (a_{ij})_{i,j=2}^{n-1} \in \mathbb{R}^{(n-2)\times(n-2)}.$$

Furthermore, we set the matrix

$$\mathbb{A}' := \begin{pmatrix} A' & 0 \\ 0 & A' \end{pmatrix} \in \mathbb{R}^{2(n-2)\times 2(n-2)}$$

△

I.e., A' emerges from A by truncating the first and last row and column. To emphasize the original roles of the entries of A' we let its indices run from 2 to $n-1$, instead of from 1 to $n-2$.

Lemma 3.1.15: *Let $\mathcal{G}_A$ be an elastic, nailed and strict Gaussian filter. It acts on $\mathfrak{N}'$ via $s'_{sep} \mapsto \mathbb{A}' s'_{sep} + B$ for a suitable $B \in \mathbb{R}^{2(n-2)}$.*

Proof. In order to see why this holds, we start by giving a function putting $\mathfrak{N}'$ and $\mathfrak{S}$ in relation. We define

$$\begin{aligned} \Xi : \mathfrak{N}' &\to \mathfrak{S}, \\ (x_2, ..., x_{n-1}, y_2, ..., y_{n-1})^T &\mapsto (S_x, x_2, ..., x_{n-1}, E_x, S_y, y_2, ..., y_{n-1}, E_y)^T. \end{aligned}$$

I.e., we just add the fixed start and end points back to our truncated list of points and keep track of the order of x- and y-coordinates. It is clear that Ξ is a bijection between $\mathfrak{N}'$ and $\mathfrak{N}$. In particular we can invert it on $\mathfrak{N}$. And since $\mathcal{G}_A$ is nailed, it operates on $\mathfrak{N}'$ via

$$s'_{sep} \mapsto \Xi^{-1}\left(\mathcal{G}_A(\Xi(s'_{sep}))\right).$$

We know how $\mathcal{G}_A$ acts on the element $\Xi(s'_{sep})$: First, the entries S_x, E_x, S_y, E_y stay the same, as $\mathcal{G}_A$ is nailed. Second, an x-coordinate x_i gets changed into

$$a_{i,1}S_x + \sum_{j=2}^{n-1} a_{ij}x_j + a_{i,n}E_x \tag{3.3}$$

and an y-coordinate y_i into

$$a_{i,1}S_y + \sum_{j=2}^{n-1} a_{ij}y_j + a_{i,n}E_y. \tag{3.4}$$

Applying Ξ^{-1} then truncates the coordinates of S, E again, resulting in an element of $\mathfrak{N}'$. When we set

$$\begin{aligned} b_i^x &:= a_{i,1}S_x + a_{i,n}E_x, \\ b_i^y &:= a_{i,1}S_y + a_{i,n}E_y \end{aligned}$$

and, consequently,

$$B := (b_2^x, ..., b_{n-1}^x, b_2^y, ..., b_{n-1}^y)^T,$$

the above formulae (3.3) and (3.4) for the entries of $\Xi^{-1}\left(\mathcal{G}_A(\Xi(s'_{sep}))\right)$ can be merged to the single term $\mathbb{A}'s'_{sep} + B$, which is what we wanted to show. □

Now we only need a few more steps to understand this operation.

Lemma 3.1.16: *Given an elastic, nailed and strict Gaussian filter $\mathcal{G}_A$. Operating on $\mathfrak{N}'$, it is a contraction with respect to the Euclidean norm on $\mathfrak{N}' = \mathbb{R}^{2(n-2)}$.*

Proof. Due to A being elastic, the first and last row of A' have a row-sum smaller than 1. To be precise, the second line of A has to have a non-zero entry at the first position and the $(n-1)$-st at the last entry. And they get cut off in the transition from A to A'.

Because of that, we can apply Perron-Frobenius Theory (to be precise, Theorem 1.1 and Corollary 1 in [58]) to get that the largest eigenvalue of A' is strictly smaller than 1. This then clearly holds for $\mathbb{A}'$, too.

Finally, the Spectral Theorem implies that the operator norm of the map $s'_{sep} \mapsto \mathbb{A}' s'_{sep}$ is equal to this largest eigenvalue and is, hence, also smaller than 1. This directly implies that $s'_{sep} \mapsto \mathbb{A}' s'_{sep}$ is a contraction. Concatenating this linear map with any translation will produce an affine map $s'_{sep} \mapsto \mathbb{A}' s'_{sep} + B$ which is still a contraction. As Lemma 3.1.15 states, $\mathcal{G}_A$ operating on $\mathfrak{N}'$ is exactly of this form.

$\square$

Corollary 3.1.17: *An elastic, nailed and strict Gaussian filter $\mathcal{G}_A$ is also a contraction on $\mathfrak{N}$.*

This leads to the final result of this section.

Theorem 3.1.18: *Let $\mathcal{G}_A$ be an elastic, nailed and strict Gaussian filter and $s \in \mathfrak{S}$ be any stroke. Then the iterated application of $\mathcal{G}_A$ to s converges towards a straight line segment u on the line connecting the start and end points P_1, P_n of s.*

Proof. This follows simply from Banach's Fixed-Point Theorem: Corollary 3.1.17 tells us that $\mathcal{G}_A$ is is a contraction on $\mathfrak{N}_{P_1,P_n}$. Therefore, a unique fixed stroke of $\mathcal{G}_A$ exists and the series $\left(\mathcal{G}_A^k(s)\right)_{k=1}^{\infty}$ converges to it. We then conclude using Proposition 3.1.10 which assures that his fixed stroke is a line segment. $\square$

In the beginning of this section we mentioned that iterating Gaussian blur on a grey-value picture will result in an image that consists of a single grey value. It is plausible to call such an image "featureless". If we transfer this interpretation to strokes, Theorem 3.1.18 implies that straight line segments are featureless strokes. This matches Section 3.1.2 in which we argued that straight lines are featureless as they eliminate all degrees of freedom of any feature.

Finally, we want to briefly mention the invertibility of Gaussian filters:

Whenever a Gaussian filter $\mathcal{G}_A$ is invertible, the inverse is given by A^{-1}, and it is automatically a Gaussian filter, too. This holds, because $A\mathbf{1} = \mathbf{1}$ directly implies $\mathbf{1} = A^{-1}\mathbf{1}$.

The problem is that the inverse of a strict Gaussian filter is only strict again when it is a permutation matrix — which is a well-known fact from the theory of stochastic matrices. This can be observed geometrically, too, when we apply a similar argument as in the proof of Proposition 3.1.10: The points of the image stroke under a Gaussian filter always lie inside the convex hull of the original stroke points. So, applying the inverse means that the image points will lie outside the convex hull of the initial stroke. So, applying an inverse Gaussian filter will blow up any stroke. This can be used to exaggerate properties of strokes — similar to affine combinations of strokes — but one has to be careful as the divergence rate is quite high.

3.1.5 Many more...

Of course, there are many, many other transformations that are "geometric" in the sense that they change the shape of strokes in a meaningful and predictable way. The four classes presented above can be described mathematically in great detail — other transformations are much more heuristic in nature. To illustrate this, we give two quick examples.

First, we can cut off the first/last part of any stroke. We already encountered this as the essential part of Algorithm 2.3.2 for hook removal and we will see this in Example 3.2.3. Here, we can drop the first and last $\lfloor 0.1 \cdot n \rceil$ points and then re-sample the remaining stroke to get back to the sample rate of n.

Second, we can add random noise to every stroke. I.e., we replace a point P_i by $P_i + \varepsilon \cdot (\cos(\alpha), \sin(\alpha))$ with both $\alpha \in [0, 2\pi]$ and $\varepsilon > 0$ arbitrary. We will use this later in Chapter 5.

A final note on stroke deformations in general: We can easily define a metric on the stroke space $\mathfrak{N}$ via

$$d(s,t) := \max_{i=1,\ldots,n} \|P_i - Q_i\|_2 \,.$$

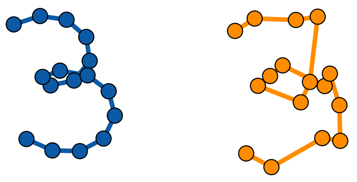


Figure 3.13: *Adding noise to every point of a stroke.*

Then we could define any bijective map $T : \mathfrak{N} \to \mathfrak{N}$ to be a **reasonable** transformation as long as $d(s, T(s)) < \varepsilon$ for an appropriate $\varepsilon > 0$ and all $s \in \mathfrak{N}$. However, two strokes with a relatively large distance, with respect to the metric above, might still look quite similar; especially if they represent a well-known character. E.g., in Figures 3.2, 3.4, 3.7, 3.12 and 3.9 we see that individual points P_i move far away from their original position even though the overall semantic of the stroke does not get lost.

With this in mind, it seems to be more sensible to not use arbitrary perturbations but structured deformations like the ones presented in this chapter.

3.2 Applications

The transformations introduced above usually have a direct practical use. For example,

— Gaussian filters are used for noise reduction as we alluded to already several times,

— projective transformations describe perspective distortion which should not affect HWR algorithms and

— convex combinations are the most direct way to morph strokes into one another.

If they are combined, however, more areas of applications emerge. First, we can combine Gaussian filters and convex combinations to create a smoothing map that, in contrast to Gaussian filters alone, does not converge towards a straight line segment but stays close in shape to the original stroke.

Second, we can create more training data by simply applying a select set of transformations to an existing data set several.

Third, when we want to computer-generate a text that looks like written by hand, we can use these transformations to create natural-looking variance. This can be applied, for example, in chat programs: a user gives a small handwriting sample, and the program then generates screen output that looks as if handwritten by the user.[9]

We will now illustrate these applications in short examples.

Example 3.2.1: Let $\mathcal{G}$ be a minimal Gaussian filter. And for any stroke s let u be straight line segment connecting P_1 and P_n of s and which is fixed by $\mathcal{G}$. From Theorem 3.1.18 we know that $\left(\mathcal{G}^k(s)\right)_{k=0}^{\infty}$ converges towards u. So, sooner or later, the stroke $\mathcal{G}^k(s)$ will not be recognisable as s anymore. However, it will get smoother and smoother. In order to keep the transformed stroke as close to s as possible but still get a similar smoothing effect, there is an obvious idea:

[9]Of course, for this thesis we ignore all safety concerns and whether something like this is welcomed.

apply $\mathcal{G}$ only a few times. Here we propose another approach which does not degenerate the stroke and still achieves "maximal smoothness".

Let α be the parameter of the Gaussian filter, i.e., $\mathcal{G} = \mathcal{G}_{A_\alpha}$, and let $\rho \in (0,1)$ be another constant. Define

$$\begin{aligned} s^0 &:= s, \\ s^k &:= \rho \cdot \mathcal{G}\left(s^{k-1}\right) + (1-\rho) \cdot s \qquad k \geq 1. \end{aligned}$$

That means that we contract and smooth the stroke via the Gaussian filter $\mathcal{G}$, but then we push the resulting stroke back towards the original one. For $\rho = 1$ we get the familiar sequence which converges. But for $\rho < 1$ there exists a limit, too. This, again, follows from Banach's Fixed Point Theorem and the proof from Lemma 3.1.15 can directly be adapted to show the contraction property. We will sketch this here briefly:

Denote the map $t \mapsto \rho \cdot \mathcal{G}(t) + (1-\rho) \cdot s$ by $\mathcal{E}_{\alpha,\rho,s}$. I.e., we see the stroke s as a parameter of this map. It acts on truncated strokes $\mathfrak{N}'$ via $t'_{sep} \mapsto \rho \mathbb{A}' t'_{sep} + \tilde{B}$ for a suitable $\tilde{B} \in \mathbb{R}^{2(n-2)}$. Just as in the lemma we compute $\mathcal{E}_{\alpha,\rho,s}(\Xi(t'_{sep}))$. Its x- and y-coordinates, respectively, are

$$\begin{aligned} &\rho \cdot \left(a_{i,1} S_x + \sum_{j=2}^{n-1} a_{ij} x_j + a_{i,n} E_x \right) + (1-\rho) \cdot c_{i,x} \quad \text{and} \\ &\rho \cdot \left(a_{i,1} S_y + \sum_{j=2}^{n-1} a_{ij} y_j + a_{i,n} E_y \right) + (1-\rho) \cdot c_{i,y}, \end{aligned}$$

where we denoted the coordinates of s by $c_{i,x}$ and $c_{i,y}$ to emphasise that they are constants here. We set

$$\begin{aligned} \tilde{b}_i^x &:= \rho \cdot (a_{i,1} S_x + a_{i,n} E_x) + (1-\rho) \cdot c_{i,x}, \\ \tilde{b}_i^y &:= \rho \cdot \left(a_{i,1} S_y + a_{i,n} E_y\right) + (1-\rho) \cdot c_{i,y} \end{aligned}$$

and

$$\tilde{B} = (\tilde{b}_2^x, \ldots, \tilde{b}_{n-1}^x, \tilde{b}_2^y, \ldots, \tilde{b}_{n-1}^y)^T.$$

Then we can write $\mathcal{E}_{\alpha,\rho,s}$ as an affine map $t'_{sep} \mapsto \rho \mathbb{A}' t'_{sep} + \tilde{B}$. And since $|\rho| \leq 1$

the eigenvalues of $\rho\mathbb{A}'$ are also smaller than 1. So, $\mathcal{E}_{\alpha,\rho,s}$ is a contraction.

Knowing this, the sequence $\left(s^k\right)_{k=0}^{\infty}$ does converge, but the limit stroke will depend on the initial s. The larger ρ, the closer this limit stroke will be to the straight line segment; and the smaller ρ, the closer the limit is to the original s. When we combine a relatively small ρ with a large α, the limit will both look very similar to the initial stroke and be very smooth.

For example, we can set $\rho = 0.85$ and $\alpha = 0.3$. The limits for a few different highly irregular starting strokes s under $\mathcal{E}_{0.3,0.85,s}$ are illustrated in Figure 3.14. And Widget A.5 can be used to produce more examples.

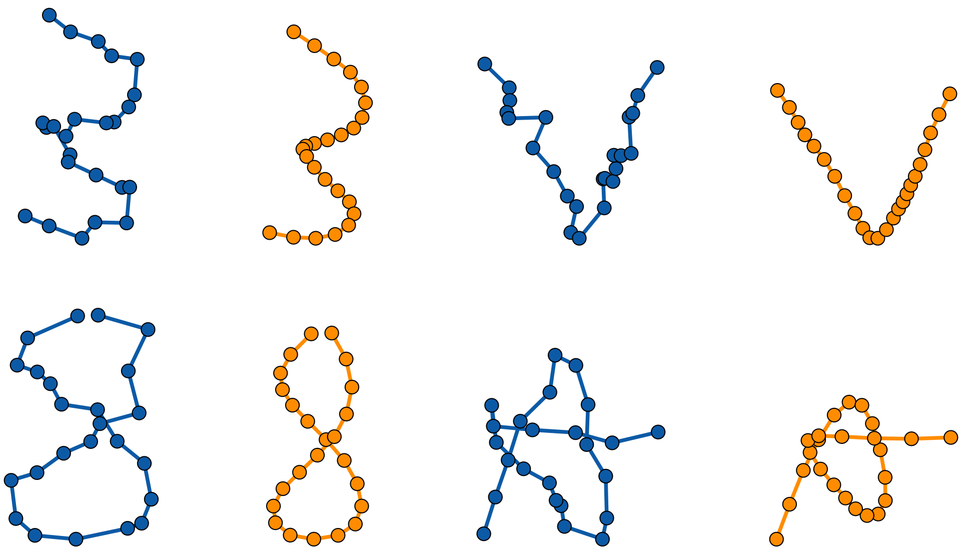


Figure 3.14: *The limit of s^k for $k \to \infty$ approximated by s^{1000} with $\rho = 0.85$ and $\alpha = 0.3$.*

As we can see, most local irregularities are removed, but the overall appearance is preserved. Comparing this effect with regular Gaussian filters, via Widgets A.4 and A.5, reveals that the same transformed strokes appear in both cases. The difference, however, is that they are just one of many intermediate results when applying Gaussian filters alone. Combining them with affine pulls creates the strokes shown in the figure as the end result.

Still, the deformation might be relatively strong depending on the initial stroke. In general, ρ and α have to be balanced to produce a reasonable result. Also, the choice for both parameters depends on the sample rate n.

We get a variation of this if we push the intermediate result even further away from the straight line segment. We set

$$\begin{aligned} &s^0 := s, \\ &s^k := \rho \cdot \mathcal{G}(s^{k-1}) + (1-\rho)\cdot(2s-u) \qquad \forall k \geq 1. \end{aligned}$$

So, the second summand of the convex combination is given by an affine combination of s and u. Here, concretely, $2s - u$. With the interpretation of the straight line segment u being featureless, $2s - u$ can be seen as the stroke whose features are twice as pronounced as the ones of s. (Recall Section 3.1.2.) And as $\mathcal{G}$ eliminates features, the hope is that this construction has a limit that matches s qualitatively much more. △

Example 3.2.2: Consider the set S of thirty strokes representing a 3 in Figure 3.15. We can multiply them by applying any combination of any number of transformations introduced in this chapter. We can do this is in two similar, but slightly different ways.

Let F be a feature vector that, hopefully, describes the strokes we are interested in well. Here, in this example, we take

$$F : \Omega \to [0,1]^3, \quad s \mapsto \begin{pmatrix} f_6(s) \\ f_{14}(s) \\ f_5(s) \end{pmatrix}$$

from Section 2.3.3. They are the STRAIGHTNESS, the x-coordinate of the START POINT and STROKE RETURN. Moreover, let T be a transformation of strokes that we want to use. Here we will take $T = \mathcal{G}^5 \circ R$ as the concatenation of a rotation R by $\frac{2\pi}{360}$ to the right and of a minimal Gaussian filter $\mathcal{G} = \mathcal{G}_{A_{\frac{1}{20}}}$ that maps

$$P_i \mapsto \frac{1}{20}P_{i-1} + \frac{18}{20}P_i + \frac{1}{20}P_{i+1}.$$

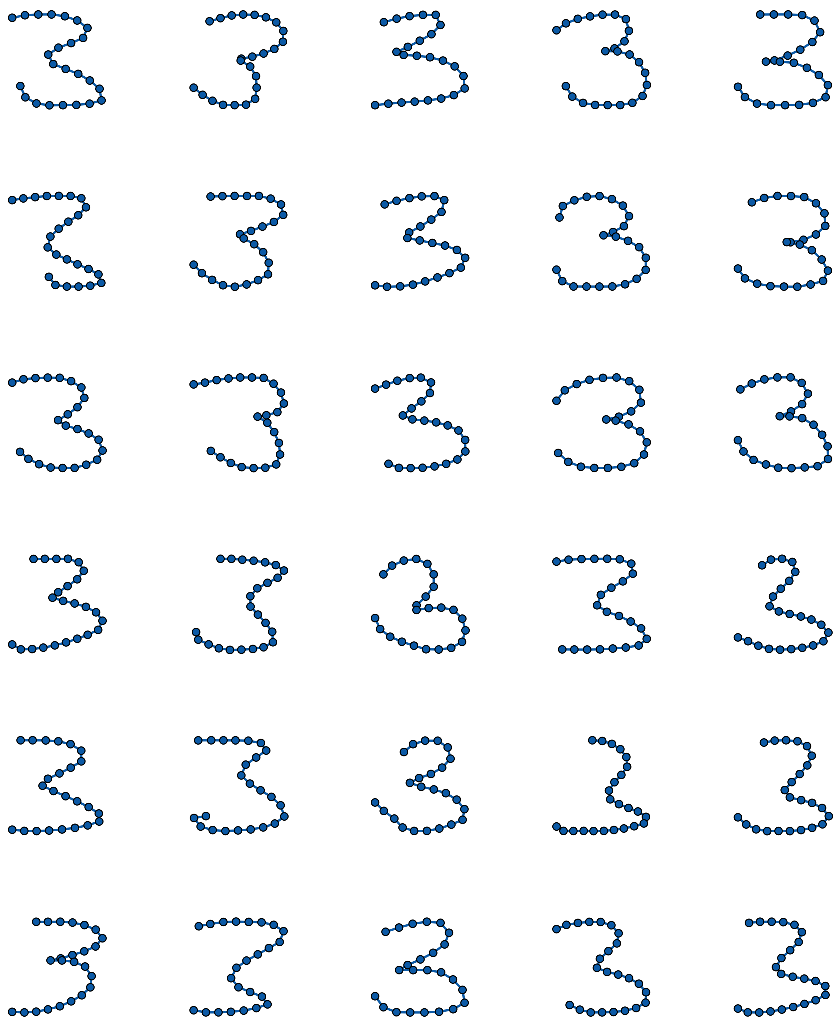


Figure 3.15: *"Raw" data depicting variants of 3.*

Note that the STRAIGHTNESS and STROKE RETURN are invariant under rotations, but not under Gaussian filters; and that the START POINT is invariant under (nailed) Gaussian filters, but not under rotations. Since STRAIGHTNESS and STROKE RETURN describe the form of the stroke and the START POINT its position, this fits the qualitative interpretation of Gaussian filters and projective transformations.

With a chosen confidence level $\alpha = 0.1$ we can perform a Kolmogorov-Smirnov test for each feature f_i comparing $f_i(S)$ and $f_i\left(T^k(S)\right)$ for integers $k > 1$. I.e., we want to find out how often the transformation T can be applied to our sample data until the distribution of the feature values changes significantly. We define k_i to be the largest integer k such that $f_i(T(S)), ..., f_i\left(T^k(S)\right)$ are still distributed the same way as $f_i(S)$.

For the concrete strokes, features and transformation above we get the following values:

$$\begin{aligned} k_6 &= 7 \\ k_{14} &= 8 \\ k_5 &= 14 \end{aligned}$$

To preserve the innate 3-ness of the strokes under T we cannot apply it more than 7 times — at least if we assume that the given features describe this 3-ness properly. Applying T now 7 times to S results in thirty new strokes. We see that all of them clearly still represent a 3; at least as much the strokes in Figure 3.15 do.

To "thicken" the set of 3-samples even more, we can compute arbitrarily many convex combinations of these now sixty strokes. Say, we only compute the "mid-strokes", i.e., $\frac{1}{2}s + \frac{1}{2}t$ for s, t being either an original stroke or one that was transformed seven times. This results in a total of $60 + \binom{60}{2} = 1830$ samples for the character 3. This method, especially if convex combinations are included, creates many new strokes. In general, we can get from $|S|$ to

$$2 \cdot |S| + \binom{2 \cdot |S|}{2} = 2 \cdot |S|^2 + |S|$$

strokes if we use only a single transformation and the one form of convex combinations given. And the additional stroke are very close to the original ones.

Alternatively, we can perform the same basic steps, but use it in a much more exploratory way: Again, we consider iterations of the transformation T applied to the samples in S, compare their feature values via the Kolmogorov-Smirnov test and compute the values k_i. However, instead of the minimal value, we take the maximum value plus one: 15. This number means that T^{15} completely changes the inherent 3-ness of the strokes — again based on the assumption that the given features describe it well.

Now, a human user can compare the strokes in $T^{15}(S)$ to the ones in S and

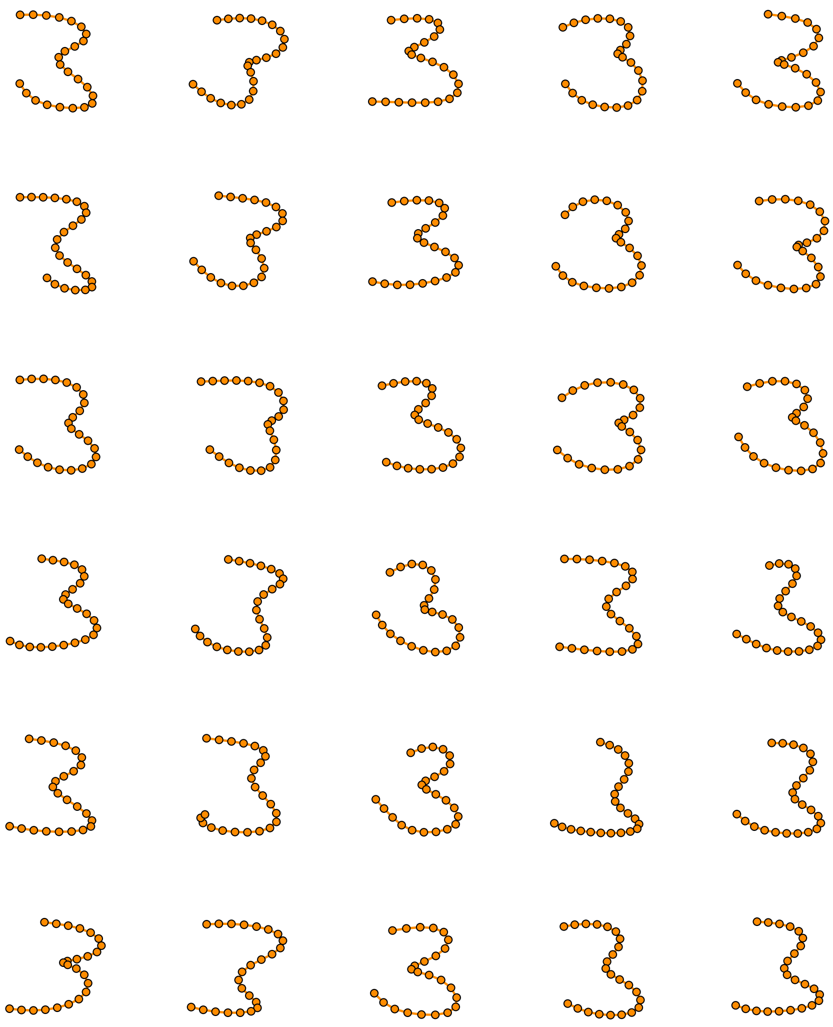


Figure 3.16: *Samples for* 3 *transformed by* T^7*.*

judge whether they can still be considered a, in this case, 3. If so, we can replace S by $S \cup T^{15}(S)$ and start over again. This allows for a systematic exploration of the stroke space to find more good samples. However, in contrast to the first method, here the new strokes have significance difference to the originals.

When we look at the elements of $T^{15}(S)$ depicted in Figure 3.17, we see that most are still recognisable as 3's. But some are already borderline cases; especially the stroke in the fifth row, fourth column. So, it might be justifiable to include all these new samples in a training data set, but it is definitely not as obvious as with $T^7(S)$.

Moreover, since looking at $T^{15}(S)$ can take a person very long for large sample sets S, one can instead only present a relatively small subset $S' \subset T^{15}(S)$ to the user. Then, adding S' to S is reliable, but slow. Adding $T^{15}(S)$ entirely, even

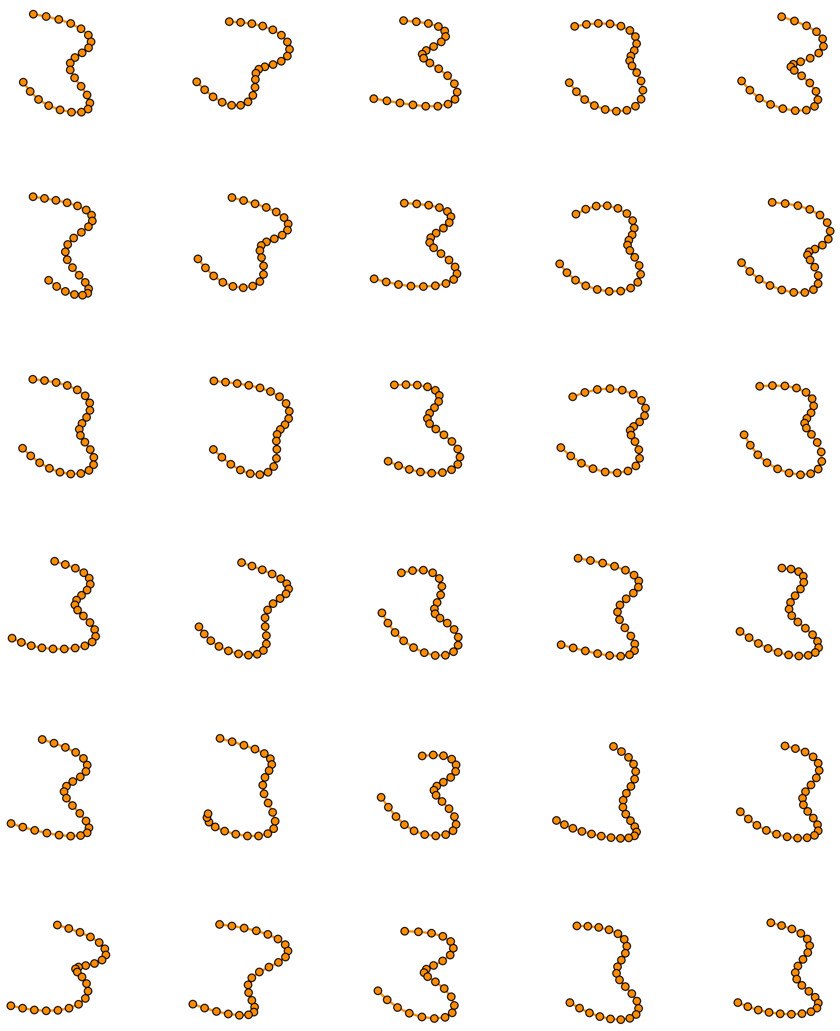


Figure 3.17: *Samples of 3 transformed by* T^{15}.

though only S' was checked, is fast, but unreliable. Moreover, it needs stochastic analyses to be justified.

A final note: To allow for more nuance in both methods, one can consider the second smallest (or m-th smallest) k-value in the first one and the second largest (or m-th largest) in the second one (for heuristically or systematically defined m).

See Chapter 6 for how this sample multiplication process is employed in ALICE:HWR. $\triangle$

Example 3.2.3: Imagine a chat program for smart phones that allows the user to type some text which then gets converted into strokes that imitate the handwriting of the user. This combines the speed of typing with the personality of handwritten words. Ideally, the conversion process introduces variations into the strokes such that not all characters look the same. However, the overall style and appearance of the strokes should not change.

Here in this example we want to illustrate that this can be achieved with the transformations presented above. We use a very short text and transform every stroke the same way.[10] Moreover, we want to use signatures as this short text, because they usually carry more "personality" than normal text. In particular, we consider longhand here explicitly as a possible input.

To be a bit more flexible than in the theoretical considerations so far, we sample every stroke s with $\left\lceil \frac{\text{Length}(s)}{C} \right\rceil$ points for a fixed constant C.

Below, we will use three different types of transformations: a simplification map given by a combination of Gaussian filters and accelerations, cutting off portions of start and end of each stroke and a distortion given by a projective transformation. They imitate the process of

— writing faster with less details,

— putting the finger/pen down too late and lifting it up to early and

— writing at a different angle with a different hand position.

Concretely:

— Let $\mathcal{G} = \mathcal{G}_{A_{\frac{1}{20}}}$ be the minimal Gaussian filter with parameter $\frac{1}{20}$.

— Let $A = A_{0.001}$ be the acceleration by 0.001.

— Let C_S and C_E be the transformations that cut off the first and last point of a stroke, respectively, and then re-samples the remaining stroke to have n points again.

[10]To make a longer text look both consistent and varied one has to put in more work than just transforming every stroke the same way.

— Let M be the projective transformation given by

$$M = \begin{pmatrix} \cos\left(-\frac{1}{8}\pi\right) & -\sin\left(-\frac{1}{8}\pi\right) & 0 \\ \sin\left(-\frac{1}{8}\pi\right) & \cos\left(-\frac{1}{8}\pi\right) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \frac{6}{\sqrt{31}} \begin{pmatrix} 1 & \frac{1}{12} & 0 \\ -\frac{1}{3} & \frac{5}{6} & 0 \\ 5 & \frac{2}{3} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & \frac{3}{10} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

This projective transformation combines a shearing parallel to the x-axis to the right with a "complicated" transformation containing a perspective distortion and a rotation to the right. The pre-factor $\frac{6}{\sqrt{31}}$ is used to factor out the determinant of the transformation in the middle. This has the effect that M does not change the size of the strokes.

Now, given any recorded stroke r, we build the transformed stroke t by

$$t = 0.375 \cdot \left(\frac{1}{2}\mathcal{G}^5(r) + \frac{1}{2}A^5(r)\right) + 0.529 \cdot \left(\frac{1}{2}C_S(r) + \frac{1}{2}C_E(r)\right) + 0.096 \cdot M(r).$$

All transformations are added up via a convex combination. This allows for continuous mixing of all effects, which can be tested in Widget A.6. The concrete weights here are chosen heuristically and give a nice looking result.

Two peculiarities to notice: Firstly, applying a minimal Gaussian filter or an acceleration k times has a qualitatively similar effect as multiplying their respective parameters by k. The fact that we apply them here five times in an artefact from searching for an appealing combination of transformations. Secondly, taking the average of $C_S r$ and $C_E r$ is, of course, different from cutting off both ends at once and leads to a more "interesting" effect.

The result of this whole duplication process can be seen in Figure 3.18, where the process is iterated three times in total. We see that — even without analysing the shape of the stroke and making inferences on how to preserve the concrete shapes — we can create variations of human handwriting that look convincing.

As already mentioned in the beginning of this example, applying the same transformations the same number of times to each stroke is probably not the best method. But Figure 3.18 and Widget A.6 illustrate that this is already a good and simple way to preserve characteristics of handwriting while changing the actual strokes.

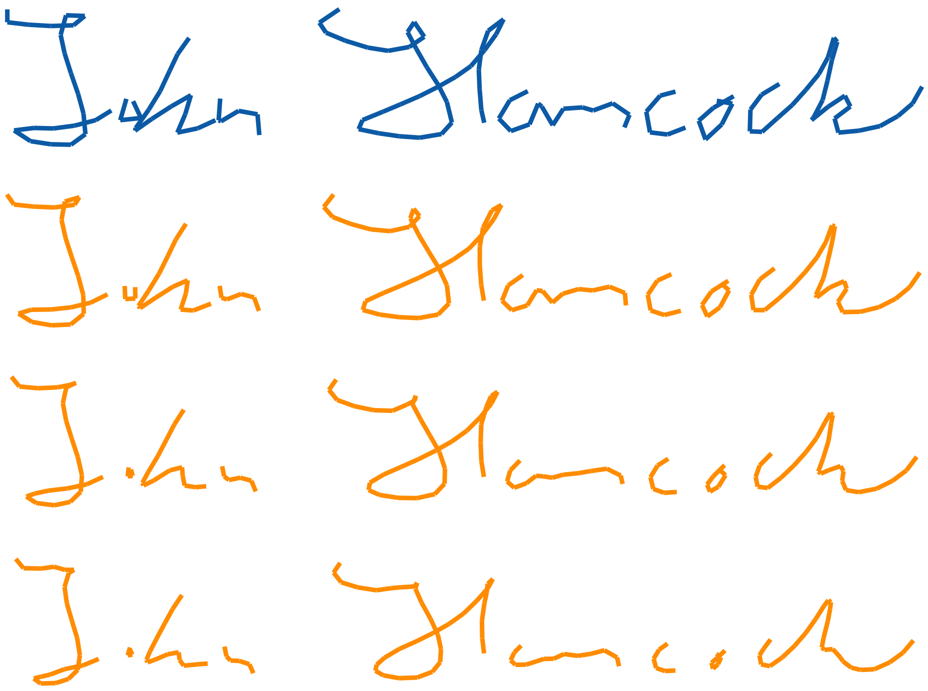


Figure 3.18: *Duplicating a signature with variations.*

And there are, of course, many more possibilities to alter strokes such that the typeface does not change too much. In particular, when more samples of the same characters and words are available. △

The examples above illustrate how deforming strokes can be useful both for HWR directly and for other applications. Moreover, we saw that the transformations we defined all preserve the overall shape of strokes. We will makes this more stringent in Chapter 5. For now we are content with the fact that we can create new samples for stroke types from old ones; which we used to build and judge classification steps for ALICE:HWR.

4 Aspects of stroke classification

Ten percent of nothin' is... let me do the math here... nothin' into nothin'... carry the nothin'...

— Jayne Cobb, *Firefly*

In this chapter, we will present and discuss particular points in the classification process used in ALICE:HWR. The results in this chapter are formulated in a theoretical setting with certain assumptions. However, we will also show how these results manifest in practical examples.

As the early builds of ALICE:HWR were the motivation to explore the topics of this chapter, the version used in the ALICE iBook does not fully accord with the way they are presented. We talk about this in Chapter 5.

4.1 Directional vectors

The core classification step in ALICE:HWR is to sort every stroke into a few large categories. Some of them are given by features 1, 2, 3, 4, 12, 13, 22 and 23 as given in Section 2.3.3. They describe the direction of certain parts of the stroke. We compute them by considering normalised vectors between points on a stroke and projecting them onto directions we are interested in. Afterwards, types that match the recorded stroke the least with respect to these directional vectors are excluded.[1]

For example, consider the alphabet $\mathbb{A} = \{\mathtt{S}, \mathtt{2}, \mathtt{3}\}$ and compare the Start Direction. If written from top to bottom, strokes representing `2` and `3` start with a pen/finger move towards the right while `S` starts towards the left. So, when a recorded stroke r starts towards the right we can classify it as not-S. Afterwards, the classifier would do something else to decide whether r is a `2` or `3`.

Here we want to explain why and how good this exclusion step works. We formulate the results in terms of general directional vectors — i.e., points on the unit circle. The prime example will be the Start Direction, but much more unintuitive computations are possible as well — e.g., the normalised vector from the top left corner of the bounding box to the point on the stroke with the lowest y-coordinate.

For the statements below, we denote the error function by erf. I.e.,

$$\operatorname{erf}(x) := \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

Theorem 4.1.1: *Let $\mathbb{A}$ be an alphabet and for any stroke $s \in \mathfrak{S}$ let v_s be a point on the unit circle. Assume that the position of v_s for a stroke s representing $l \in \mathbb{A}$ is given by the truncated normal distribution* $\operatorname{TrN}(C_l, d_l)$

— *centred at a point C_l on the unit circle with standard deviation d_l and*

— *truncated at the point opposite C_l.*

[1]For more general exclusion rules given by arbitrary features see Section 4.2.

Moreover, assume that none of the intervals along the unit circle with centre C_l and width $2d_l$ overlap. Both these assumptions are illustrated in Figure 4.1. Denote

$$M := \max_{l \in \mathbb{A}} d_l \quad \textit{and} \quad m := \min_{l \in \mathbb{A}} d_l.$$

Let r be a recorded stroke and E an integer between 1 and $|\mathbb{A}|$. Furthermore let $\mathbb{E} \subset \mathbb{A}$ be the set of E stroke types which are farthest away from v_r, and let $l_0 \in \mathbb{A}$ be the type r actually represents. Then, the probability that $l_0 \in \mathbb{E}$, i.e., that C_{l_0} is one of the E far-away centres from v_r, is bounded from above by

$$P\left(l_0 \in \mathbb{E} \;\middle|\; r \textit{ represents } l_0\right) \leq 1 - \frac{\operatorname{erf}\left(\frac{(2|\mathbb{A}|-2E+1)\cdot m - M}{2\sqrt{2}\cdot M}\right)}{\operatorname{erf}\left(\frac{\pi}{\sqrt{2}\cdot m}\right)}.$$

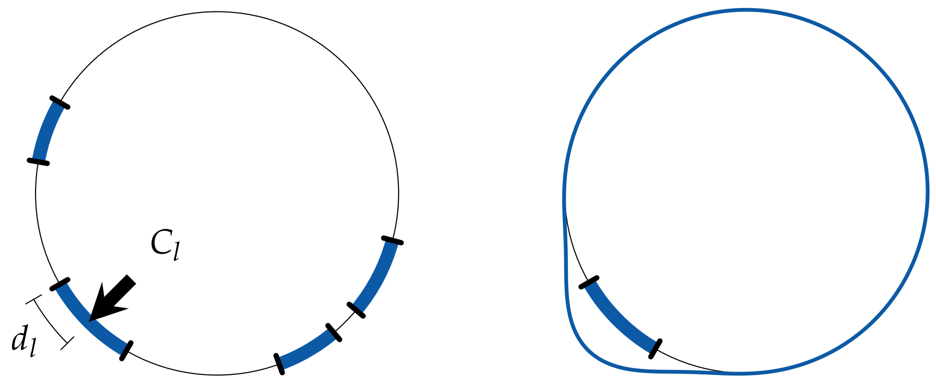


Figure 4.1: *Several non-overlapping intervals along the unit circle with centres C_l and widths $2d_l$ (left) and a qualitative depiction of the truncated normal distribution associated to one of them (right).*

Corollary 4.1.2: *In the situation of Theorem 4.1.1, assume that all d_l are equal to the same value d. Then the upper bound becomes*

$$1 - \frac{\operatorname{erf}\left(\frac{|\mathbb{A}|-E}{\sqrt{2}}\right)}{\operatorname{erf}\left(\frac{\pi}{\sqrt{2}\cdot d}\right)}.$$

We will prove the theorem, but the corollary is what we want to have in mind when using the statement in praxis: When we exclude certain stroke types based on directional vectors associated to the samples, we might exclude the type the recorded stroke represents. The probability that this happens is bounded by the given term. Also, we see directly that this bound is higher the more types we want to exclude and lower the more the samples are clustered.

Proof. The cumulative distribution function F of a normal distribution with mean μ and standard deviation σ that is truncated outside the interval $[a, b]$ is given by

$$F(x) = \frac{\operatorname{erf}\left(\frac{x-\mu}{\sqrt{2}\cdot\sigma}\right) - \operatorname{erf}\left(\frac{a-\mu}{\sqrt{2}\cdot\sigma}\right)}{\operatorname{erf}\left(\frac{b-\mu}{\sqrt{2}\cdot\sigma}\right) - \operatorname{erf}\left(\frac{a-\mu}{\sqrt{2}\cdot\sigma}\right)}.$$

When we parametrise the unit circle by the interval $[-\pi, \pi]$ such that C_{l_0} sits at 0 we get $\mu = 0$ and so, for our case, we obtain

$$F(x) = \frac{\operatorname{erf}\left(\frac{x}{\sqrt{2}\cdot d_{l_0}}\right) - \operatorname{erf}\left(\frac{-\pi}{\sqrt{2}\cdot d_{l_0}}\right)}{\operatorname{erf}\left(\frac{\pi}{\sqrt{2}\cdot d_{l_0}}\right) - \operatorname{erf}\left(\frac{-\pi}{\sqrt{2}\cdot d_{l_0}}\right)}.$$

Using the antisymmetry of the error function, this simplifies to

$$F(x) = \frac{\operatorname{erf}\left(\frac{x}{\sqrt{2}\cdot d_{l_0}}\right) + \operatorname{erf}\left(\frac{\pi}{\sqrt{2}\cdot d_{l_0}}\right)}{2\cdot\operatorname{erf}\left(\frac{\pi}{\sqrt{2}\cdot d_{l_0}}\right)} = \frac{1}{2} + \frac{1}{2}\cdot\frac{\operatorname{erf}\left(\frac{x}{\sqrt{2}\cdot d_{l_0}}\right)}{\operatorname{erf}\left(\frac{\pi}{\sqrt{2}\cdot d_{l_0}}\right)}.$$

The probability that $d_{circ}(v_r, C_{l_0})$ is equal or larger than a positive value x is then

$$\begin{aligned} P\left(d_{circ}(v_r, C_{l_0}) \geq x \;\middle|\; r \text{ represents } l_0\right) &= F(-x) + (1 - F(x)) \\ &= 2\cdot F(-x) \\ &= 1 - \frac{\operatorname{erf}\left(\frac{x}{\sqrt{2}\cdot d_{l_0}}\right)}{\operatorname{erf}\left(\frac{\pi}{\sqrt{2}\cdot d_{l_0}}\right)} \\ &\leq 1 - \frac{\operatorname{erf}\left(\frac{x}{\sqrt{2}\cdot M}\right)}{\operatorname{erf}\left(\frac{\pi}{\sqrt{2}\cdot m}\right)}, \end{aligned}$$

where the first equality holds due to the symmetry of F. What is left to do is bound this distance x from below in the case that C_{l_0} is among the E centres farthest ways from v_r.

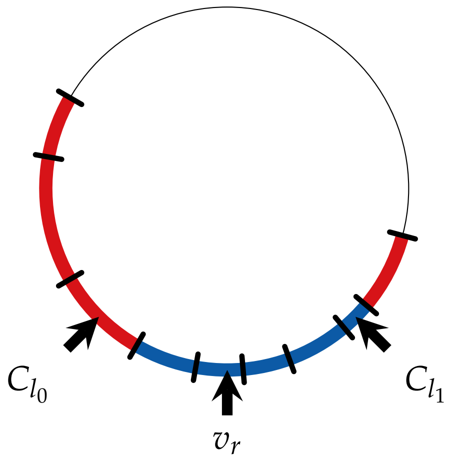


Figure 4.2: *The minimal distance of from C_{l_0} to C_{l_1} along the unit circle. (Intervals associated to lines in $\mathbb{E}$ are red, all others are blue.)*

As we are only interested in the minimal distance x between v_r and C_{l_0}, we can assume that all the intervals touch. There is, w.l.o.g., a closest centre among $\{C_l \mid l \in \mathbb{E}\}$ to v_r. We assume that this is C_{l_0}, again because we want to minimise $d_{circ}(v_r, C_{l_0})$. Let C_{l_1} be the the farthest centre away from v_r such that $l_1 \notin \mathbb{E}$ and C_{l_1} lies on the other side than C_{l_0}. See Figure 4.2 for an illustration of this situation. Then the distance y from C_{l_0} to C_{l_1} along the arc which contains v_r is

$$\begin{aligned} y &= d_{l_0} + \sum_{l \in \mathbb{A} \setminus \mathbb{E}} 2d_l - d_{l_1} \\ &\geq \min_{l \in \mathbb{E}} d_l + (|\mathbb{A}| - E) \cdot \min_{l \in \mathbb{A} \setminus \mathbb{E}} 2d_l - \max_{l \in \mathbb{A} \setminus \mathbb{E}} d_l \\ &\geq (2\,|\mathbb{A}| - 2E + 1) \cdot \min_{l \in \mathbb{A}} d_l - \max_{l \in \mathbb{A}} d_l \\ &= (2\,|\mathbb{A}| - 2E + 1) \cdot m - M. \end{aligned}$$

This total distance y is split in two (in general unequal) parts by v_r. In order for C_{l_0} to be farther away from v_r than C_{l_1}, the minimal distance x we search has to be bigger than both of the these two parts. As the bigger part will always be at least half of the total, we end up with

$$d_{circ}(v_r, C_{l_0}) \ \geq \ x \ \geq \ \frac{1}{2}y \ \geq \ \frac{1}{2}\left((2\,|\mathbb{A}| - 2E + 1) \cdot m - M\right).$$

With the formula for the probability computed above, we finally get

$$\begin{aligned} & P\left(l_0 \in \mathbb{E} \ \middle| \ r \text{ represents } l_0\right) \\ \leq\ & P\left(d_{circ}(v_r, C_{l_0}) \geq \frac{1}{2}\left((2\,|\mathbb{A}| - 2E + 1) \cdot m - M\right) \ \middle| \ r \text{ represents } l_0\right) \\ \leq\ & 1 - \frac{\operatorname{erf}\left(\frac{(2|\mathbb{A}|-2E+1)\cdot m - M}{2\sqrt{2}\cdot M}\right)}{\operatorname{erf}\left(\frac{\pi}{\sqrt{2}\cdot m}\right)}. \end{aligned}$$

□

Example 4.1.3: Assume we have an alphabet with four stroke types

$$\mathbb{A} = \{\texttt{blue, orange, green, violet}\}$$

and a collection of good samples $\mathfrak{T}$. For all these strokes s we look at their start directions $v_s = \frac{P_3 - P_1}{\|P_3 - P_1\|_2}$ and mark them on the unit circle. More concretely, imagine

— `blue` encodes strokes representing 2 and 3,

— `orange` encodes δ and g,

— `green` encodes 6, and

— `violet` encodes the brackets) and }.

Then, reasonably well-written samples for these stroke types have the start directions shown by Figure 4.3.

If the sets $\{v_s \mid s \in \mathfrak{T}_l\}$ of start directions look similar as in the figure, we can assign each stroke type to a quadrant on the circle depending on where the good samples for this line lie. We mark each midpoint of the quadrants as the centre associated to a line and can then classify a recorded stroke r via its start direction v_r.

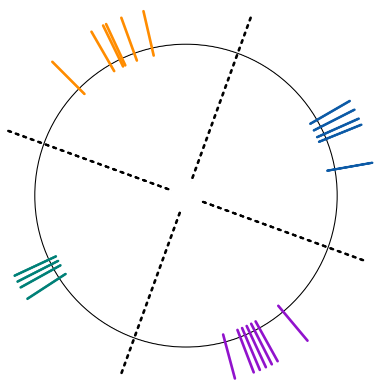


Figure 4.3: *Start directions of good samples of four lines aggregating in four quadrants (left).*

The simplest way to do this would be to assume that r represents the type l if the centre C_l is closest do v_r. However, because of fluctuations in the writing process, this might be too strong a statement. Instead, we look at the centre l_0 which is farthest from v_r and exclude l_0 form $\mathbb{A}$ for any subsequent analysis. The probability that we make a mistake — i.e., that C_{l_0} being far away from v_r despite r actually representing l_0 — is exactly the probability considered in Theorem 4.1.1 for $E = 1$. So, we can bound this error probability from above to estimate the reliability of this exclusion step.

We can use a statistical test that checks samples for normal distribution; but for such small sample sizes as shown here in Figure 4.3 it is hard arguing against the supposed normal distribution.

So, we assume that we can apply Theorem 4.1.1, possibly after a statistical analysis of the collection of good samples. Then we have to find the actual values d_l to evaluate the given formula for the upper bound on the error probability. For the sake of simplicity, we assume that they are all equal to the same value d, so that we can use Corollary 4.1.2.

If this d is maximal (see Figure 4.4, left), it is $d = \frac{2\pi}{2\cdot|\mathbb{A}|} = \frac{\pi}{4}$. Then the probability that we make a mistake, i.e., that we exclude the actual line r represents, is less than

$$1 - \frac{\operatorname{erf}\left(\frac{|\mathbb{A}|-1}{\sqrt{2}}\right)}{\operatorname{erf}\left(\frac{\pi}{\sqrt{2}\cdot\frac{1}{8}\cdot 2\pi}\right)} = 1 - \frac{\operatorname{erf}\left(\frac{3}{\sqrt{2}}\right)}{\operatorname{erf}\left(\frac{4}{\sqrt{2}}\right)} \approx 0.00026366.$$

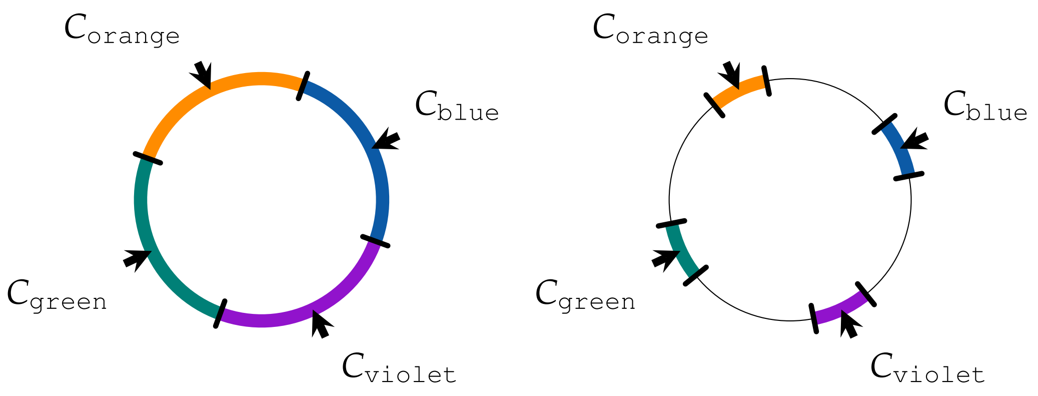


Figure 4.4: *Maximal (left) and relatively small (right) associated intervals/standard deviation.*

If we instead assume a d that is much smaller and approximately half the range the samples of each line lie in (see Figure 4.4, right), we get $d = \frac{2\pi}{32} = \frac{\pi}{16}$ and an error probability of

$$1 - \frac{\operatorname{erf}\left(\frac{3}{\sqrt{2}}\right)}{\operatorname{erf}\left(\frac{16}{\sqrt{2}}\right)} \approx 0.00026998.$$

In both cases the probability is less than 0.03% so we can safely assume that excluding the line whose centre is farthest way from the recorded v_r does not accidentally excludes the correct line. Moreover, we see that the exact value of d only makes a minor difference.

Now consider the same situation with eight different stroke types whose start directions point into eight different octants — for example the "shafts" of the arrows ↑↗→↘↓↙←↖. Then the standard deviations d_l of the samples can be assumed to be equal due to the symmetry of the symbol sets. Again assuming they are maximal, we get $d = \frac{2\pi}{2 \cdot |\mathbb{A}|} = \frac{\pi}{8}$, which leads to an error probability of at most

$$1 - \frac{\operatorname{erf}\left(\frac{7}{\sqrt{2}}\right)}{\operatorname{erf}\left(\frac{8}{\sqrt{2}}\right)} \approx 2.558 \cdot 10^{-12}.$$

Unsurprisingly, when we have a "large" number of types we want to analyse, and when they can be distinguished solely by their start direction, excluding the line which fits the recorded data the worst gives a wrong result with a very low probability.

We can make the same computations with a larger E, say, $E := \frac{|\mathbb{A}|}{2}$. I.e., we bisect the set of potential stroke types into two halves: the $\frac{|\mathbb{A}|}{2}$ best matches to the recorded stroke with respect to the start direction and the $\frac{|\mathbb{A}|}{2}$ worst matches. And we assume, again, that the standard deviations are equal and maximal.

In the first example with $\mathbb{A} = \{\texttt{blue}, \texttt{orange}, \texttt{green}, \texttt{violet}\}$, this means we exclude the worst two matches and make a mistake in doing so with a probability less than

$$1 - \frac{\operatorname{erf}\left(\frac{2}{\sqrt{2}}\right)}{\operatorname{erf}\left(\frac{4}{\sqrt{2}}\right)} \approx 0.0454.$$

In the second example with $\mathbb{A} = \{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow\}$, this means we exclude the worst 4 matches and make a mistake in doing so with a probability less than

$$1 - \frac{\operatorname{erf}\left(\frac{4}{\sqrt{2}}\right)}{\operatorname{erf}\left(\frac{8}{\sqrt{2}}\right)} \approx 0.0000633.$$

So, excluding the worse half and continuing with the better half might be a mistake every 25th if we only have four stroke types. But with eight types it is already viable.

If we directly classify by taking the single best match — i.e., when we exclude $E = |\mathbb{A}| - 1$ stroke types — the error rate might be as high as

$$1 - \frac{\operatorname{erf}\left(\frac{1}{\sqrt{2}}\right)}{\operatorname{erf}\left(\frac{4}{\sqrt{2}}\right)} \approx 1 - \frac{\operatorname{erf}\left(\frac{1}{\sqrt{2}}\right)}{\operatorname{erf}\left(\frac{8}{\sqrt{2}}\right)} \approx 0.317.$$

So, this cannot be expected to work in general. △

This example illustrates that if we find directional vectors that describe subsets of the characters we want to recognise well, we can use them reliably as a (first) classification step.

When they are well chosen, we can cut the possible symbols the recorded stroke represents in half. Moreover, excluding the worst match is almost always a very reliable heuristic; especially for large $|\mathbb{A}|$.

4.2 Exclusion rules via FCA hypotheses

In Section 4.1 we looked at exclusion rules built from features that describe some form of orientation of strokes. It can help a lot in restricting the potential stroke types a recorded stroke might represent. In general, however, we might not have enough of such orientation features that cluster as well as we want to. Therefore, we present a more general method, based on Formal Concept Analysis, to find exclusion rules that is applicable for all features.

For this section let $\mathbb{A}$ be an alphabet, $\mathfrak{T}$ a collection of good samples for $\mathbb{A}$ and $f_1, ..., f_m$ a set of features. Consider the (multi-valued) training context

$$\mathcal{T} = \left(\bigcup_{l \in \mathbb{A}} \mathfrak{T}_l, \;\; \mathbb{A} \cup \{f_1, ..., f_m\}, \;\; [0,1], \;\; I \right)$$

as defined in Section 2.2.2 and build the (ordinary) context $\mathcal{T}_F$ from it by

1. deleting all columns associated to the stroke type $l \in \mathbb{A}$ — i.e., forgetting which type a sample stroke represents — and
2. scaling it ordinally: the attributes obtained from the feature values shall be parametrised by (f_i, y) for a feature f_i and a value $y \in [0, 1]$ such that a stroke/object s has attribute (f_i, y) if and only if $f_i(s) \geq y$.

In particular, if a stroke/object s has attribute (f_i, y_0) it also has all attributes (f_i, y) with $y \leq y_0$. And the largest value y for which a stroke/object has attribute (f_i, y) is $y = f_i(s)$.

Remark 4.2.1: Note that we use a scale that leads to a context with an infinite attribute set. For the theoretical considerations here this will be no problem. In practice, we subdivide the interval $[0, 1]$ into disjoint bins into which we sort the feature values. See Example 2.1.23. $\triangle$

Lemma 4.2.2:

1. *The intents of $\mathcal{T}_F$, except for $\{f_1, ..., f_m\} \times [0,1]$, are of the form*

$$\left\{ (f_i, y) \;\middle|\; 0 \leq y \leq y_i \quad s.t. \quad \exists s_i \in \bigcup_{l \in \mathbb{A}} \mathfrak{T}_l : f_i(s_i) = y_i \right\}.$$

2. *They are parametrised by the set*

$$\left\{ (f_1(s_1), ..., f_m(s_m))^T \;\middle|\; s_1, ..., s_m \in \bigcup_{l \in \mathbb{A}} \mathfrak{T}_l \right\}.$$

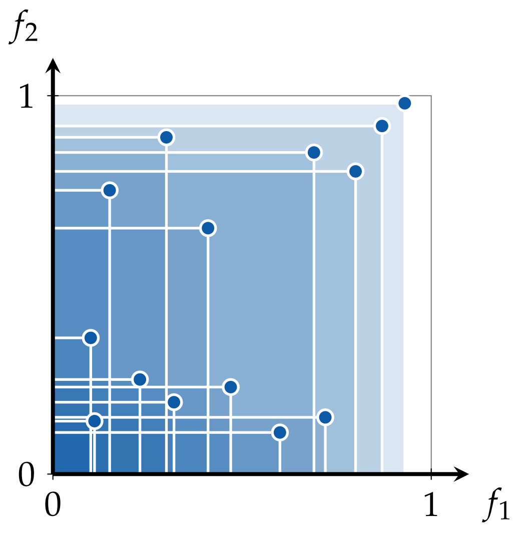


Figure 4.5: *The object intents of $\mathcal{T}_F$ and their intersections.*

Note that if we consider the feature vector

$$F : \mathfrak{S} \to [0,1]^m, \;\; s \mapsto (f_1(s), ..., f_m(s))^T,$$

these sets describe (the context-ified version of) sub-cuboids $\bigtimes_{i=1}^{m} [0, y_i]$ of the codomain of f. See Figure 4.5.

Proof.

1. By the construction of $\mathcal{T}_F$, its object intents are given by the axes-parallel sub-cuboids of $[0,1]^m$ with $\mathbf{0}$ as one corner and $F(s)$ as the opposite corner for a stroke/object $s \in \bigcup_{l \in \mathbb{A}} \mathfrak{T}_l$. The intents of a context are intersections of object intents — which, in our case, are thus represented by the intersection of these sub-cuboids. As all of them have the zero vector as a corner, it is also part of every intent. The opposite corner of such an intersection has to lie on the border of the object intents used. These borders are hyper-planes in $[0,1]^m \subset \mathbb{R}^m$ given by the equation

   $$x_i = f_i(s_i) \quad \text{or} \quad x_i = 0.$$

   Here, x_i denotes the i-th coordinate of a general point in $[0,1]^m$ and s_i is a suitable stroke in $\bigcup_{l \in \mathbb{A}} \mathfrak{T}_l$. As visualised by Figure 4.5, the right-hand hyperplane equation is never relevant unless $f_i(s_i) = 0$, but then the equations coincide.

2. The opposite corner of the zero vector in such a sub-cuboid is of the form $(f_1(s_1), ..., f_m(s_m))^T$ and it characterises it uniquely.

All these considerations can be done for all intents except the whole attribute set. It is $\{f_1, ..., f_m\} \times [0,1]$ and is represented by the whole hypercube $[0,1]^m$ it self. Unless there exists a stroke/object s with $F(s) = \mathbf{1}$, its extent is empty. □

Definition 4.2.3: Let $\mathcal{C} = (G, M, I)$ be a context. Assume that we have a subset of objects $A \subset G$ we want to recognise by some attributes. We think of A as **positive examples** for a virtual/unknown attribute and of $G \backslash A$ as **negative examples**. Let

$$\mathcal{C}^+ := (A, M, I^+) \quad \text{with} \quad I^+ := I \cap (A \times M) \quad \text{and}$$
$$\mathcal{C}^- := (G \backslash A, M, I^-) \quad \text{with} \quad I^- := I \cap ((G \backslash A) \times M).$$

I.e., they are sub-contexts emerging form $\mathcal{C}$ by restricting to the relevant rows. Denote their derivation operator by $(.)^+$ and $(.)^-$, respectively.

A **(+)-hypothesis**[2] for A is an intent of $\mathcal{C}^+$ that is not present in any negative example. I.e., it is a set $h \subset M$ such tha

$$h^{++} = h \quad \text{and} \quad h' \cap (G \backslash A) = \emptyset.$$

△

Proposition 4.2.4: *Given a set of strokes/objects $S \subset \bigcup_{l \in \mathbb{A}} \mathfrak{T}_l$, let $(f_1(s_1), ..., f_m(s_m))^T$ be the tuple uniquely describing S' for adequate $s_1, ..., s_m \in S$. Then the following hold:*

1. *A (+)-hypothesis for S exists if and only if for all $s \in (\bigcup_{l \in \mathbb{A}} \mathfrak{T}_l) \backslash S$ there exists an $i \in \{1, ..., m\}$ with $f_i(s) < f_i(s_i)$.*

2. *The (practical) decision rule derived from this (+)-hypothesis is given by the (axes-parallel) sub-cuboid spanned by $(f_1(s_1), ..., f_m(s_m))^T$ and the one vector $\mathbf{1}$. I.e., we decide that a recorded stroke r fits the sample set S if $F(r)$ lies in this sub-cuboid. And this sub-cuboid is the minimal sub-cuboid which*

   — *has the one vector $\mathbf{1}$ as a corner and*

   — *contains $F(S)$.*

Proof.

1. By definition, a (+)-hypothesis is an intent of $\mathcal{T}_F^+$ that is not contained in any object intent of strokes/objects in $(\bigcup_{l \in \mathbb{A}} \mathfrak{T}_l) \backslash S$. We get the smallest intent S' of $\mathcal{T}_F^+$ by intersecting all object intents for strokes/objects in S. With the interpretation from Lemma 4.2.2, this means we intersect all sub-cuboids of $[0,1]^m$ that are spanned by $\mathbf{0}$ and a $F(s)$ for $s \in S$. Again by Lemma 4.2.2 it is given by its corner $(f_1(s_1), ..., f_m(s_m))^T$ opposite $\mathbf{0}$; with $s_i \in S$ for all i.

   Now let $t \in (\bigcup_{l \in \mathbb{A}} \mathfrak{T}_l) \backslash S$. Its intent contains S' if and only if

$$F(t) \geq \begin{pmatrix} f_1(s_1) \\ \vdots \\ f_m(s_m) \end{pmatrix}$$

[2]pronounced "positive hypothesis"

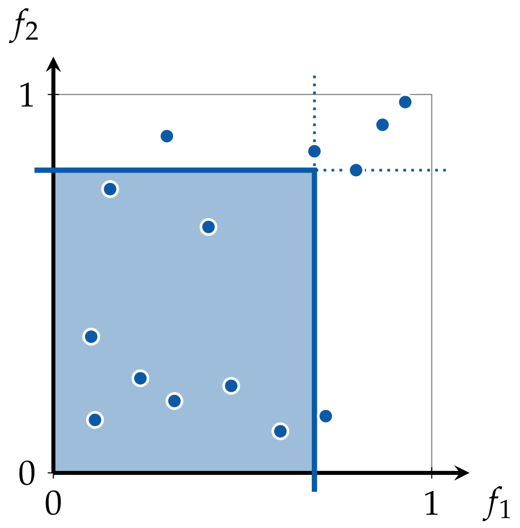


Figure 4.6: *A (+)-hypothesis (left) and its associated decision rule (right); with $F(S)$ being given by the four dots in the top right corner.*

component-wise. So, its intent does not contain S' if one of these inequalities are violated.

2. The decision rule associated to a (+)-hypothesis is that a recorded stroke/object $F(r)$ belongs to S if its intent contains the (+)-hypothesis. With the same argument from part 1, this means that

$$F(r) \geq \begin{pmatrix} f_1(s_1) \\ \vdots \\ f_m(s_m) \end{pmatrix}.$$

And the set of all points in $[0,1]^m$ that fulfil this is exactly the sub-cuboid spanned by $(f_1(s_1), ..., f_m(s_m))$ and $\mathbf{1}$.

□

The situation of Proposition 4.2.4 is illustrated in Figure 4.6. (+)-hypotheses are sub-cuboids at the "bottom left" of the $[0,1]^m$ cube. But strokes/objects whose intents contain them have to have feature vectors "in the upper right" of them. So, the classification area is just the complementary sub-cuboid.

Note that in part 2 of the proposition we built only one decision rule, and we used the (unique) minimal (+)-hypothesis for it. Any other (+)-hypothesis contains it and, thus, the "decision cuboid" at the "top right" becomes smaller.

The practical summary of the last two statements is the following: If we scale the training context such that high feature values imply low feature values, hypotheses gained from formal concept analysis describe when all feature values are large enough to classify an unknown stroke/object. We can use Proposition 4.2.4 directly for some classification. However, it becomes more versatile when we remember how to build complements of fuzzy sets and how to use different scales.[3]

Example 4.2.5: Consider the alphabet

$$\mathbb{A} = \{\texttt{blue, orange, green, violet, red}\}.$$

As in Section 4.1, we want to build exclusion rules. I.e., we want to decide when a recorded stroke r does not fit some of these stroke types instead of deciding when it does.

Assume that we have a collection of good samples, and two features f_1, f_2 such that the feature vectors of these samples are the following:

`blue`	$\begin{pmatrix}6.0\\1.1\end{pmatrix}, \begin{pmatrix}7.2\\1.5\end{pmatrix}$
`orange`	$\begin{pmatrix}1.5\\7.5\end{pmatrix}, \begin{pmatrix}3.0\\8.9\end{pmatrix}$
`green`	$\begin{pmatrix}1.0\\3.6\end{pmatrix}, \begin{pmatrix}1.1\\1.4\end{pmatrix}, \begin{pmatrix}2.3\\2.5\end{pmatrix}, \begin{pmatrix}3.2\\1.9\end{pmatrix}$
`violet`	$\begin{pmatrix}6.9\\8.5\end{pmatrix}, \begin{pmatrix}8.0\\8.0\end{pmatrix}, \begin{pmatrix}8.7\\9.2\end{pmatrix}, \begin{pmatrix}9.3\\9.8\end{pmatrix}$
`red`	$\begin{pmatrix}4.1\\6.5\end{pmatrix}, \begin{pmatrix}4.7\\2.3\end{pmatrix}$

They are depicted in Figure 4.7. Apart from the appropriate colours, `blue` points are squares, `orange` points are diamonds, `green` points are triangles, `violet` points are upside-down triangles and `red` points are circles. Now we

[3]For our application here, these two things are the same.

want to find decision rules for when recorded feature vectors do not match `red` ones which “cluster” in the centre of $[0,1]^2$.

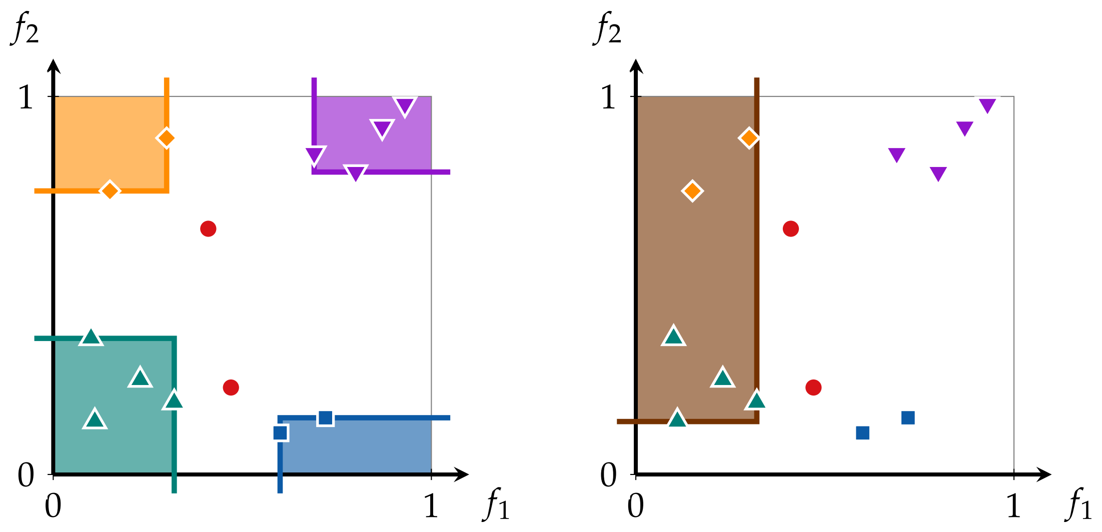


Figure 4.7: *Four decision rules for* NOT-RED *based on single stroke types (left) and a decision rule based on several types (right).*

In terms of hypotheses, this means we consider the goal attribute NOT-RED. So, we label all sample strokes of the type `red` as RED and all sample strokes of the other four types as NOT-RED. If we scale the training context $\mathcal{T}_F$ as described at the beginning of this section — larger feature values imply low feature values — we can apply Proposition 4.2.4 with the set $S = \mathfrak{T}_{\texttt{violet}}$ being all samples of type `violet`. The (+)-hypothesis S' has top-right corner $\begin{pmatrix} 6.9 \\ 8.0 \end{pmatrix}$. That means that

$$\textit{a recorded stroke } r \textit{ is } \text{NOT-RED} \textit{ if } f_1(r) \geq 6.9 \textit{ and } f_2(r) \geq 8.0.$$

Now we can scale $\mathcal{T}$ differently. We replace all values $f_1(s)$ in $\mathcal{T}$ by $1 - f_1(s)$. I.e., we consider the fuzzy set that describes the complementary property to whatever f_1 describes. When we then scale $\mathcal{T}$ ordinally again as in the beginning, large $1 - f_1$ values will imply low ones. In other words, low f_1 values imply large ones. We could have achieved this directly by scaling the attributes of $\mathcal{T}$ in exactly this way.

When we now apply Lemma 4.2.2 and Proposition 4.2.4, the (+)-hypotheses for NOT-RED are located at the bottom right and the decision rules at the top left. So, it is reasonable to apply them to $S = \mathfrak{T}_{\texttt{orange}}$. Then, the concrete decision rule is:

A recorded stroke r is NOT-RED *if* $f_1(r) \leq 3.0$ *and* $f_2(r) \geq 7.5$.

When we scale $\mathcal{T}$ such that for both features low values imply high ones we can take $S = \mathfrak{T}_{\texttt{green}}$. And when we scale it such that high f_1-values imply low ones, but low f_2-values imply high ones, $S = \mathfrak{T}_{\texttt{blue}}$ is a good choice. The decision rules derived from these choices are:

A recorded stroke r is NOT-RED *if* $f_1(r) \leq 3.2$ *and* $f_2(r) \leq 3.6$.

A recorded stroke r is NOT-RED *if* $f_1(r) \geq 6.0$ *and* $f_2(r) \leq 1.5$.

The four rules we found for the different scalings of $\mathcal{T}$ and the different choices of S can be seen on the left of Figure 4.7.

The choice to choose exactly the strokes of a single type as S was, of course, arbitrary. We could very well take, for example, $S = \mathfrak{T}_{\texttt{green}} \cup \mathfrak{T}_{\texttt{orange}}$. When we then scale $\mathcal{T}$ such that high f_1-values imply low ones and that low f_2-values imply high ones, we get the rule that

a recorded stroke r is NOT-RED *if* $f_1(r) \leq 3.2$ *and* $f_2(r) \geq 1.4$.

It can be seen at the right of Figure 4.7.

Whether a particular choice for S — and also for the particular scaling — is reasonable or not, depends, of course, on the data. Here, in this case, this new decision rule might not be well-suited to distinguish between RED and NOT-RED strokes. Looking at the decision area on the right of Figure 4.7, we see that it covers the cluster of RED feature values in the f_2 directions. That means that the f_2 values are practically irrelevant. Moreover, its right border is very close to the RED cluster, so the f_1 value might not be good guidance. $\triangle$

The introduction of (+)-hypotheses we gave above was ad-hoc to fit our purposes. To understand and appreciate it more, one has to talk about general implications as we defined them in Definition 2.1.24. Applying the idea of minimal generating sets for implications to hypotheses leads to the notion of generatives. These are subsets of (+)-hypotheses and therefore classify a broader set of potential objects having the goal attribute. ALICE:HWR uses a mix of both generatives and hypotheses, and combines them via heuristic considerations for each case. However, we will not go into details how to find/compute these generatives — see [14] and [15] for information on that.

With the ideas presented in this section, and the last one, we can exclude bad matches for a recorded stroke quite well. When we have restricted the set of potential stroke types, we need to decide which is the best match though. This is the topic of the last section of this chapter.

4.3 Fuzzy matching of feature vectors

After using exclusion rules to limit the potential matches for a recorded stroke, we compare feature vectors directly by an algorithm that is basically a 1-nearest neighbour algorithm: mapping the objects we are interested in into a metric space. Then compare the images of recorded data with the images of samples for the objects in question via the given metric. Then, the sample with the smallest distance to the recorded data is deemed the best match.

In our case of HWR, we have a feature vector $F : \mathfrak{S} \to [0,1]^m$ and we interpret the images as fuzzy values. Because of that, we do not want to choose any metric on $[0,1]^m$, but build a binary function that captures and respects this fuzziness. In particular, we want this comparison function to be a fuzzy set again; instead of a metric.

If we compare two strokes s and r by a single feature $f : \mathfrak{S} \to [0,1]$ we would like to say that both strokes agree (with respect to f) if both have this feature. So, in our standard interpretation, if

$$f(s) = 1 = f(r).$$

Because of the fuzziness of the recording and measuring process, however, we also interpret value smaller but close to 1 as a sign for the presence of the property in question. Moreover, we can encode the fact that both strokes have the feature via a suitable t-norm $\odot$. Hence, we would say that both strokes agree if $f(s) \odot f(t)$ is large. I.e., we would use the function

$$\alpha = \alpha_{\odot} : [0,1] \times [0,1] \to [0,1], \quad (x,y) \to x \odot y$$

as a fuzzy set indicating how well the strokes agree by computing $\alpha(f(s), f(r))$. The problem here, when we want to use this for classification, is that this function only gives high values if a property is present in both strokes. However, the absence of a property is in itself a feature, too. I.e., if both feature values $f(s)$ and $f(r)$ are very small, we would also say that the strokes agree. To get an affirmative measure that encapsulates both, we have to modify α to

$$\alpha' : [0,1] \times [0,1] \to [0,1], \quad (x,y) \to \max\{x \odot y, \quad (1-x) \odot (1-y)\}.$$

This definition is a bit unwieldy because there is a distinction by cases: It is easy to check (using the monotonicity of t-norms) that $x \odot y$ is the larger of the two values if and only if $x + y > 1$. This makes sense in our interpretation that for large feature values $f(s)$ and $f(r)$ their product is a measure for how well the strokes fit together.

A small caveat of this definition, however, is the following: For other t-norms than the minimum we usually have

$$\frac{1}{2} \odot \frac{1}{2} < \frac{1}{2}.$$

So, when the geometric property in question is present with a likelihood of 50%, it is much less likely that the strokes agree with respect to this property. At least when using α'. When we do not know whether a property is present — the case the fuzzy value $\frac{1}{2}$ describes — we want to be equally unsure whether s and r are a match.

This is not particularly problematic when using α' for classifying strokes. But instead of progressing from here, we want to introduce an alternative that will turn out to be smooth and more symmetric, but shows the same qualitative behaviour as α' at the same time. We achieve this by using proto-features instead of ordinary features.

Definition 4.3.1: Let $\odot$ be a t-norm. Then define the **(single-)matching function**

$$\sigma = \sigma_\odot : [0,1] \times [0,1] \to [-1,1], \quad (x,y) \mapsto \operatorname{sgn}(x) \cdot \operatorname{sgn}(y) \cdot (|x| \odot |y|).$$

△

That means, in particular, that when we use ordinary multiplication as the t-norm, we get

$$\sigma(x,y) = xy.$$

And this case is the original motivation to make this transition to proto-features: the distinction of cases is now integrated in the sign and we can simply multiply these signed fuzzy-values.

Moreover, the interpretation of this matching function is the same as above if we shift the proto-feature: Two strokes agree in having a property if $\sigma(f(s), f(r))$ is close to 1 and they agree in not having said property if $\sigma(f(s), f(r))$ is close to -1. And while σ is defined via absolute values suggesting the use of folded features, it behaves like as if we use shifted features. This illustrated by the minima and maxima in Figure 4.8.

Additionally, when $x = 0 = y$, we have $\sigma(x, y) = 0$. So, σ is more anchored in the centre of its codomain. Another advantage of σ over α is its symmetry: for any $x, y \in [-1, 1]$ we have

$$\sigma(-x, y) = -\sigma(x, y) = \sigma(x, -y).$$

That means that the measure for how well two strokes agree when both have or do not have a feature is the same for how much they disagree when on stroke has the feature and the other does not.

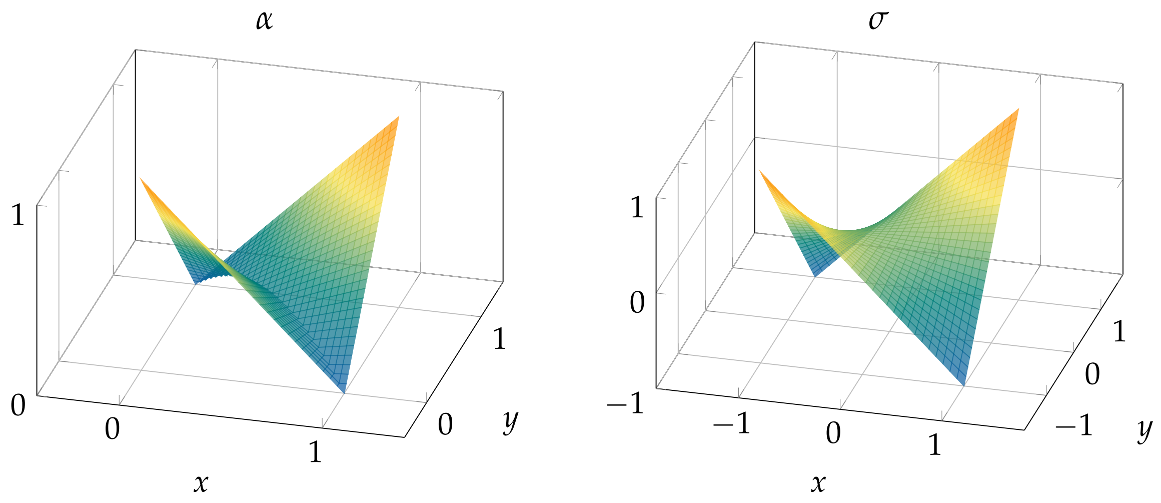


Figure 4.8: *Graphs of α and σ with multiplication as the t-norm.*

With such a matching function for a single feature it is now straightforward to define a matching function for feature vectors.

Definition 4.3.2: Let $\odot$ be a t-norm. Then define the **(multi-)matching function**

$$\Sigma = \Sigma_{\odot} : [-1,1]^m \times [-1,1]^m \to \mathbb{R}, \quad (u,v) \mapsto \sum_{i=1}^{m} \sigma_{\odot}(u_i, v_i).$$

$\triangle$

Every summand is either positive or negative, so, each entry contributes either a reward or a penalty to the overall score computed via Σ. And for the special case of $\odot$ being the ordinary multiplication we get the particularly nice formula

$$\Sigma(u,v) = u^T v.$$

Next, we want to use this function Σ as a measure of how well the features of a recorded stroke r match idealised proto-features of a given stroke type. To do so, we slightly adjust the definition of Σ. First of all, we normalise into the interval $[-1,1]$ by dividing by m. Then we can directly interpret it as proto-feature telling us how well a recorded stroke represents a certain type. Second, we only consider multiplication as the t-norm, since that is what we used in ALICE:HWR. See Section 6.2 for more on that. By abuse of notation, we will use the same symbol. So, we will work with

$$\Sigma : [-1,1] \times [-1,1] \to [-1,1], \quad (u,v) \mapsto \frac{u^T v}{m}.$$

The classification step we would do now is the following: Given an alphabet $\mathbb{A}$, a feature vector $F : \mathfrak{S} \to [-1,1]^m$, an idealised proto-feature vector F_l for each $l \in \mathbb{A}$ and a recorded stroke $r \in \mathfrak{S}$. Sort the alphabet $\mathbb{A}$ via the rule

$$l \leq l' \quad :\Longleftrightarrow \quad \Sigma(F_l, F(r)) \leq \Sigma(F_{l'}, F(r)).$$

Then we choose the largest type l as the best match for r. Or, depending on what any subsequent analysis might look like, we take the five largest stroke types. Alternatively, we take all l for which $\Sigma(F_l, F(r))$ is larger than a certain threshold.

Now we want to look at the behaviour of this matching function when $F(r)$ moves between two idealised proto-feature vectors. To do so, we make the general assumption that these

idealised proto-feature vectors only have -1 or 1 as their entries.

I.e., the proto-features we consider are built in such a way that they describe properties of perfectly drawn strokes with 100% certainty.

This assumption is motivated by the following practical consideration: When we compare two stroke types via their proto-feature vectors, an entry close to 0 means that it is not clear whether this particular feature is relevant or (a-)typical for the type in question. So, we cannot make a fair judgement whether it should be included in the comparison with another stroke type or not. Moreover, if the entry in both idealised vectors is close to zero, the impact of their product on the total matching score is minuscule.

So, let $F_l = v_0$ and $F_{l'} = v_1$ be two corners of the hypercube $[-1,1]^m$ representing two stroke types l and l'. Next we assume that they differ in k entries and that $\frac{k}{m} \leq \frac{1}{2}$. If they differ too much, we assume we have excluded one of the types anyway in the classification process before — using the method from Sections 4.1 or 4.2, or any other.

With these assumptions, we can look at a recorded proto-feature vector $F(r) = v_\lambda = (1-\lambda)v_0 + \lambda v_1$, with $\lambda \in [0,1]$ that moves from v_0 to v_1 with increasing λ. W.l.o.g., the vectors v_0, v_1 and v_λ are fo the form

$$v_0 = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ * \\ * \\ \vdots \\ * \end{pmatrix}, \quad v_1 = \begin{pmatrix} -1 \\ -1 \\ \vdots \\ -1 \\ * \\ * \\ \vdots \\ * \end{pmatrix}, \quad v_\lambda = \begin{pmatrix} 1-2\lambda \\ 1-2\lambda \\ \vdots \\ 1-2\lambda \\ * \\ * \\ \vdots \\ * \end{pmatrix}.$$

Assume that the recorded stroke r actually represents l. Then,

$$\lambda \mapsto \Sigma(v_0, v_\lambda)$$

describes how well r matches l when $F(r)$ lies closer and closer to a different idealised feature vector. It is explicitly given by

$$\Sigma(v_0, v_\lambda) = 1 - 2\frac{k}{m} \cdot \lambda =: \Sigma(\lambda).$$

Note that it only depends on $\frac{k}{m}$ and not the individual values of k and m. Its graph for various ratios $\frac{k}{m}$ can be seen in Figure 4.9.

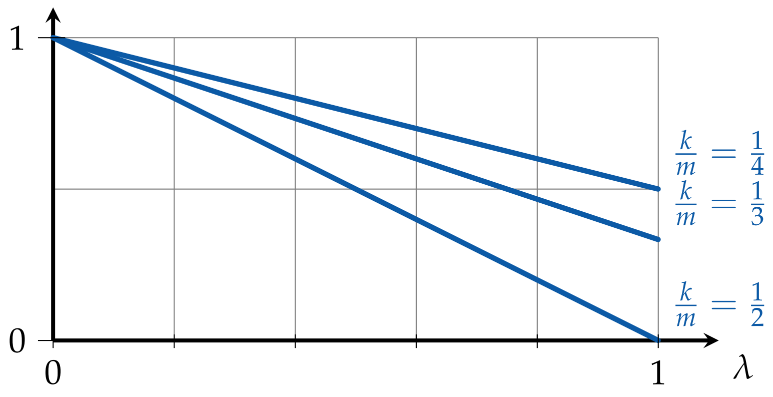


Figure 4.9: *Graphs of* $\Sigma(\lambda)$ *for various* $\frac{k}{m}$

We see that $\Sigma(1)$ is only then 0 when $\frac{k}{m} = \frac{1}{2}$. For all other values $\frac{k}{m}$, it is larger. That serves our purpose well to use Σ as a way to differentiate between stroke types: The closer $F(r)$ lies to $F_{l'} = v_1$, the smaller $\Sigma(F_l, F(r))$ becomes. But if the idealised proto-feature vectors only differ by a relatively small number of entries, the relatively large value $\Sigma(F_l, F(r))$ tells us that deciding for l' instead of l is not necessarily a safe bet; even if $F(r)$ coincides with $F_{l'}$.

The classification step just presented was used in a previous version of ALICE:HWR, with small alterations, and it works well enough in practice. But here we want to give an alternative to Σ that improves on two aspects. Both are motivated by the fact that the derivative $\frac{\partial}{\partial\lambda}\Sigma$ is constant. This means that, first, small changes in feature vectors which are close to F_l are equally severe as when $F(r)$ is close to $F_{l'}$. And since $\Sigma(\lambda)$ measures how good a match r and l are, we want this influence to be non-constant. More precise, we want small changes in $F(r)$ to matter little when $F(r)$ is close to F_l in order to have more stability in the classification process.

Second, as mentioned already above, we might not want to merely sort the stroke types $l \in \mathbb{A}$. Instead, we might want to select those for which $\Sigma(F_l, F(r))$ is larger than a certain threshold. Then, we want to have larger fuzzy values than Σ provides — to lessen the chance of accidentally excluding the correct stroke type. Moreover, finding a suitable threshold becomes easier and more reliable if $\Sigma(\lambda)$ is flat for λ close to 0, and then is steeper at one point when λ increases.

With these ideas in mind we want to find a function

$$\Pi : [-1,1] \times [-1,1] \to [-1,1]$$

that fulfils the following properties:

1. For all $\lambda \in [0,1]$ we have $\Pi(v_0, v_\lambda) \geq \Sigma(v_0, v_\lambda)$.
2. $\Pi(v_0, v_1) = \Sigma(v_0, v_1)$.
3. It is differentiable.
4. $\frac{\partial}{\partial\lambda}\Pi(v_0, v_\lambda)\Big|_0 = 0$.
5. The function $\lambda \to \Pi(v_0, v_\lambda)$ shall only depend on $\frac{k}{m}$; not on k or m individually.

Point 2 guarantees that Π and Σ show the same behaviour for large λ. This is also emphasised by point 5. And points 1, 2 and 3 together imply that there is a point in the interval $(-1,1)$ at which $\left|\frac{\partial}{\partial\lambda}\Pi(v_0, v_\lambda)\right|$ is larger than $\left|\frac{\partial}{\partial\lambda}\Sigma(v_0, v_\lambda)\right|$. Next, we show that the angle between the feature vectors provides such a function.

Lemma 4.3.3: *Define the function*

$$\Pi : [-1,1] \times [-1,1] \to [-1,1], \quad (u,v) \mapsto \frac{u^T v}{\|u\|_2 \, \|v\|_2}.$$

Then, this function Π *has all five properties listed above.*

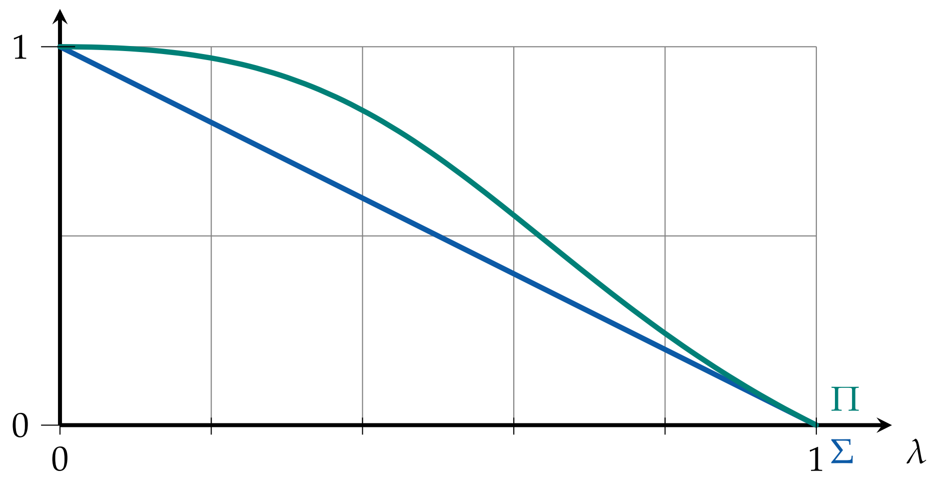


Figure 4.10: *Graphs of* $\Sigma(\lambda)$ *and* $\Pi(\lambda)$ *in the case* $\frac{k}{m} = \frac{1}{2}$.

Proof. Abbreviate $\Pi(\lambda) := \Pi(v_0, v_\lambda)$. Using the definitions of v_0 and v_λ, we can compute a function term of $\Pi(\lambda)$ explicitly. It is

$$\Pi(\lambda) = \frac{m - 2k \cdot \lambda}{\sqrt{m} \cdot \sqrt{m + 4k \cdot (\lambda^2 - \lambda)}} = \frac{1 - 2\frac{k}{m} \cdot \lambda}{\sqrt{1 + 4\frac{k}{m} \cdot (\lambda^2 - \lambda)}}.$$

This immediately proves property 5. (Note that the assumption $\frac{k}{m} \leq \frac{1}{2}$ ensures that the term under the square root is non-negative.) Property 2 is equally simple as $\Pi(1) = 1 - 2\frac{k}{m} = \Sigma(1)$. And we can prove properties 3 and 4 by computing

$$\frac{\partial}{\partial \lambda}\Pi(\lambda) = -\frac{4k(m-k) \cdot \lambda}{\sqrt{m} \cdot \sqrt{m + 4k \cdot (\lambda^2 - \lambda)}^3}$$

and

$$\left.\frac{\partial}{\partial \lambda}\Pi(\lambda)\right|_0 = 0.$$

What is left is property 1. Looking at the denominator of Π we see that $\lambda^2 - \lambda \leq 0$ for all λ. Hence, the denominator is always smaller or equal to 1. And the numerator is simply $\Sigma(\lambda)$. So, the quotient will be not smaller than $\Sigma(\lambda)$. □

Two last properties of the function Π in the context of comparing proto-feature vectors: First, comparing the angle between feature vectors means that they can be scaled by positive scalars without changing the result of Π. So, they work almost like homogeneous coordinates in the the projective space $\mathbb{RP}^{m-1}$. However, the sign of proto-feature vectors plays a role in evaluating Π. Luckily, we usually apply Π only when the vectors we want to compare are close anyway.

Second, we usually want every proto-feature vector to have at least one entry 1 and one entry -1; So we can be sure that there is a property that is definitely present and one that is definitely not. That means when we look at at a recorded proto-feature vector $F(r)$ and classify it using Π, we implicitly assume we normalised it to

$$F(r) \rightsquigarrow \frac{F(r)}{\max_{i=1,\dots,m} |f_i(r)|}.$$

So, the largest proto-feature value gets rounded to ± 1. But all other entries of $F(r)$ get scaled accordingly. This contrasts Formal Concept Analysis: if we would round entries of $F(r)$ in accordance with some implications found in the training context, each entry would be rounded individually.

Whether or not Π is a better way to sort/rank strokes types than Σ depends on the actual features and samples involved. In particular, one might be more beneficial than the other when more idealised feature-vectors per type are used to perform k-nearest neighbour classification.

The use of σ instead of α' is a promising ansatz, however, that incorporates "negative fuzzy values" (in the form of proto-features) in a sensible way. As can be seen in Section 2.3.3, these negative values often appear quite naturally in practice. And even if certain features are computed as values in $[0, 1]$, their interpretation is often more elegant as proto-features. Take, for example, Left Curvature which equals 1 if the stroke makes a left turn at every point; 0 if the stroke makes only right turns; and $\frac{1}{2}$ if the stroke makes no turns at all. Moreover, in the next chapter we will see functions which describe the shape of strokes in a coherent way and which take values in $[-1, 1]$ by default.

So, basing the classification around values in the interval $[-1, 1]$ might be a better idea than using $[0, 1]$, even though "negative fuzzy values" have not the same interpretation as normal fuzzy values.

5 Characterising strokes via determinants

It has long been an axiom of mine that the little things are infinitely the most important.
— Sherlock Holmes, *A Case of Identity*, by Arthur Conan Doyle

In Chapter 2 we introduced the most basic model for handwritten strokes. In particular, we defined the stroke space $\mathfrak{S}$ and also specific subsets like $\mathfrak{Q}$ and $\mathfrak{N}$ to normalise strokes. $\mathfrak{S}$ itself is an affine space which is used in the context of convex combinations in Section 3.1.2 and at a few other points throughout this thesis. Apart from that, there is no meaningful structure on $\mathfrak{S}$ itself that encodes what strokes are or how they behave. That means that sequences of points like the one in Figure 5.1 are in $\mathfrak{S}$. However, it is evident that it probably does not represent any real letter, digit or character in most symbol sets.

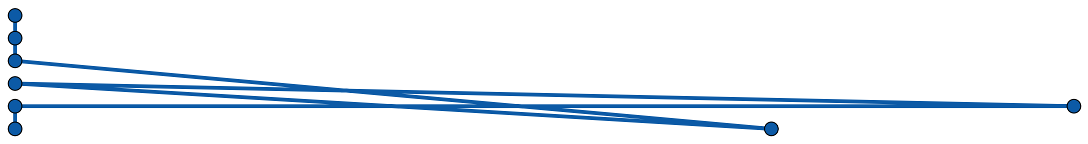


Figure 5.1: *A sequence of points that is an element of $\mathfrak{S}$, but does not represent a "real-world" stroke.*

We mentioned that we could fix this by demanding a maximal distance between neighbouring points along a line. The way we approached this problem in this thesis was by introducing the notion of sample dispersals to encode the likelihood that an element of $\mathfrak{S}$ represents something. Now, one design goal for choosing features we could demand is that they should form a decomposition of the sample dispersal maps. I.e., if $m_l : \mathfrak{S} \to [0, 1]$ models the likelihood that strokes represent a type $l \in \mathbb{A}$, a good feature set $f_1, ..., f_m : \mathfrak{S} \to [0,1]$ for this type l should (here in this scenario) fulfil

$$m_l = f_1 \odot \cdots \odot f_m$$

for an appropriate t-norm $\odot$. With the interpretation of t-norms as the intersection operator on fuzzy sets this property would mean that

a stroke represents the type l if it has all properties $f_1, ..., f_m$.

If we demanded this property in building features, we could not assume that they model any concrete geometric property. So, we did not mention sample dispersals explicitly anymore after introducing them.

Then, in Chapter 3, we looked at how various transformations act on $\mathfrak{S}$. We argued and showed in examples why these transformations are reasonable and how they can be used in practice. However, the focus of that chapter was on the practical applications: smoothing strokes without collapsing them into a line segment, creating new samples from old ones and duplicating handwritten words preserving their characteristics. Now we want to use them to analyse the general structure of strokes a bit more.

Concretely, we will, firstly, argue that looking at the curvature of a stroke helps to understand its shape. Secondly, we will describe this shape via certain determinant vectors. Lastly, we show that the transformations we know operate nicely on these vectors.

The ideas in this chapter were developed as a consequence of the results of the previous chapters and of the insights gained from the practical implementations in ALICE:HWR. Because of that, they are not yet present in any form of HWR algorithm and are indented as a starting point for further inquiries.

5.1 Curvature

In Section 4.1 we saw that classifying strokes via directional vectors works very well. In particular, these vectors include the Start Direction $\frac{P_3-P_1}{\|P_3-P_1\|_2}$, the End Direction $\frac{P_n-P_{n-2}}{\|P_n-P_{n-2}\|_2}$ and the General Direction $\frac{P_n-P_1}{\|P_n-P_1\|_2}$ which all found their way into ALICE:HWR in some way. Features like these — either given by their coordinates or their position on the unit circle — can be seen as **orientation features**. That means that they allow differentiating between rotated strokes or parts of strokes that are rotated. E.g., the can help to distinguish `V` form Λ or $\cap$ from $\cup$ or $-$ from $|$.

This leaves the question what the difference between strokes with the same orientation feature values are. Compare, for example, the letters `U`, `V` and `W` and the symbol $\cup$. Strokes representing these four characters have the same Start and End Directions and even their start and end points are at the same place.[1] The property that they do not have in common is their **shape**.

Definition 5.1.1: A (proto-)feature f on $\mathfrak{S}$ is called a **shape (proto-)feature** if it is invariant under translations and rotations. I.e., for any such transformation $T : \mathfrak{S} \to \mathfrak{S}$ and any $s \in \mathfrak{S}$ we demand $f(T(s)) = f(s)$. $\triangle$

In the rest of this chapter, we want to, firstly, argue that the (discretised) curvature of strokes is a shape feature that is actually relevant for the shape of a stroke. Secondly, we want to show that a set of certain determinants is an adequate representation of this shape.

Here we want to argue and illustrate that the curvature and the shape of a stroke have a strong interrelation and, especially, are closer related than the shape of a stroke and the actual position of its points.. For smooth curves in the plane — i.e., C^∞ maps $\gamma : [0,1] \to \mathbb{R}^2$ — the curvature is simply defined as the absolute value of the second derivative — while the second derivative itself can be thought of as the acceleration a particle experiences that travels along

[1]Start and End point are also orientation features. They allow for a similarly good classification as directional vectors. But as there are no obvious assumptions to their distribution in general, we did not analyse them separately.

the curve.[2] The discrete version of the derivative is given by the map

$$d_n : \left(\mathbb{R}^2\right)^n \to \left(\mathbb{R}^2\right)^{n-1}, \quad (P_i)_{i=1}^n \mapsto (P_{i+1} - P_i)_{i=1}^{n-1}.$$

If we now compute the second discrete derivative $(d_{n-1} \circ d_n)(s)$ of a stroke explicitly, its $(i-1)$-st entry will be

$$\kappa_i := (P_{i+1} - P_i) - (P_i - P_{i-1}) = P_{i-1} - 2P_i + P_{i+1}$$

for $i = 2, \ldots, n-1$. We index it this way instead of by $1, \ldots, n-2$ because of the following geometric interpretation: κ_i is the discrete acceleration at point P_i. If we attach κ_i as a vector to P_i, it is the diagonal of the parallelogram spanned by P_{i-1}, P_i and P_{i+1}. This elementary geometric fact is illustrated in Figure 5.2. Now, taking the absolute value of κ_i gives a measure of the discrete curvature of the stroke.

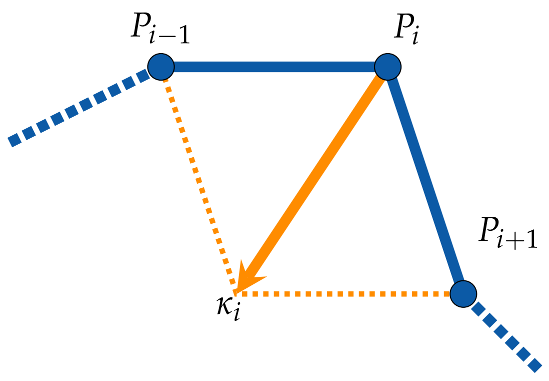


Figure 5.2: *Discrete acceleration/curvature at point P_i.*

The mathematical definition of the curvature of a stroke only makes sense if it is parametrised with respect to arc length—i.e., $\|\gamma'\|_2 = 1$. The discrete version of this given by $\|P_{i+1} - P_i\|_2$ being constant for all $i = 1, \ldots, n-1$. We will use this later when we interpret the values $\|\kappa_i\|_2$ a bit differently.

[2]We do not make a formal distinction between the acceleration as a vector and the curvature as its length. In particular, we will call both objects "curvature" in order to not confuse anything with the transformations we called accelerations.

Now we want to see how the curvature of a stroke affects its appearance. As the meaning of a stroke is ambiguous, we explore this on examples. To do so, we will randomly change the coordinates of a stroke, its first and its second derivative and compare the results qualitatively.

Consider a stroke s and add to any point P_j on it a vector $r_j e^{i\varphi_j}$ with the φ_j being uniformly chosen from the interval $[0, 2\pi)$ and the r_j from $[0, \varepsilon]$ for a fixed upper bound $\varepsilon > 0$. Here, i is the imaginary unit; not an index. Denote this operation $\left(\mathbb{R}^2\right)^n \to \left(\mathbb{R}^2\right)^n$ by $\Omega_{n,\varepsilon}$.

Adding random noise in this fashion can be tested interactively in Widget A.7. The result for one stroke representing a 3 can be seen in Figure 5.3. The top left stroke is the original, and the first column to the right of it is the result of applying $\Omega_{n,\varepsilon}$ three different times. Here, the upper bound is chosen to be 0.01.

Then, we do the same thing for the first and second derivative: we apply $\Omega_{n-1,\varepsilon} \circ d_n$ and $\Omega_{n-2,\varepsilon} \circ d_{n-1} \circ d_n$ to a stroke s. But as these are element of $\left(\mathbb{R}^2\right)^{n-1}$ and $\left(\mathbb{R}^2\right)^{n-2}$, respectively, we have to integrate them first to get back an element of $\mathfrak{S}$. We define the discrete integral as a function $\int_S^{n-1}$ with a point $S \in \mathbb{R}^2$ by

$$\int_S^{n-1} : \left(\mathbb{R}^2\right)^{n-1} \to \left(\mathbb{R}^2\right)^n, \quad (v_i)_{i=1}^{n-1} \mapsto \left(S + \sum_{j=0}^{i-1} v_j\right)_{i=1}^{n}$$

with $v_0 := 0$. This allows us to compare a stroke s with

$$\Omega_{n,\varepsilon}(s),$$
$$\left(\int_{P_1}^{n-1} \circ \Omega_{n-1,\varepsilon} \circ d_n\right)(s) \text{ and}$$
$$\left(\int_{P_1}^{n-1} \circ \int_{P_2-P_1}^{n-2} \circ \Omega_{n-2,\varepsilon} \circ d_{n-1} \circ d_n\right)(s).$$

Again, this is implemented in Widget A.7 and can be seen in Figure 5.3. Where the first column shows random noise on the points of a stroke themselves, the second column shows how noise affects the first derivative and the third column shows it for the second derivative.

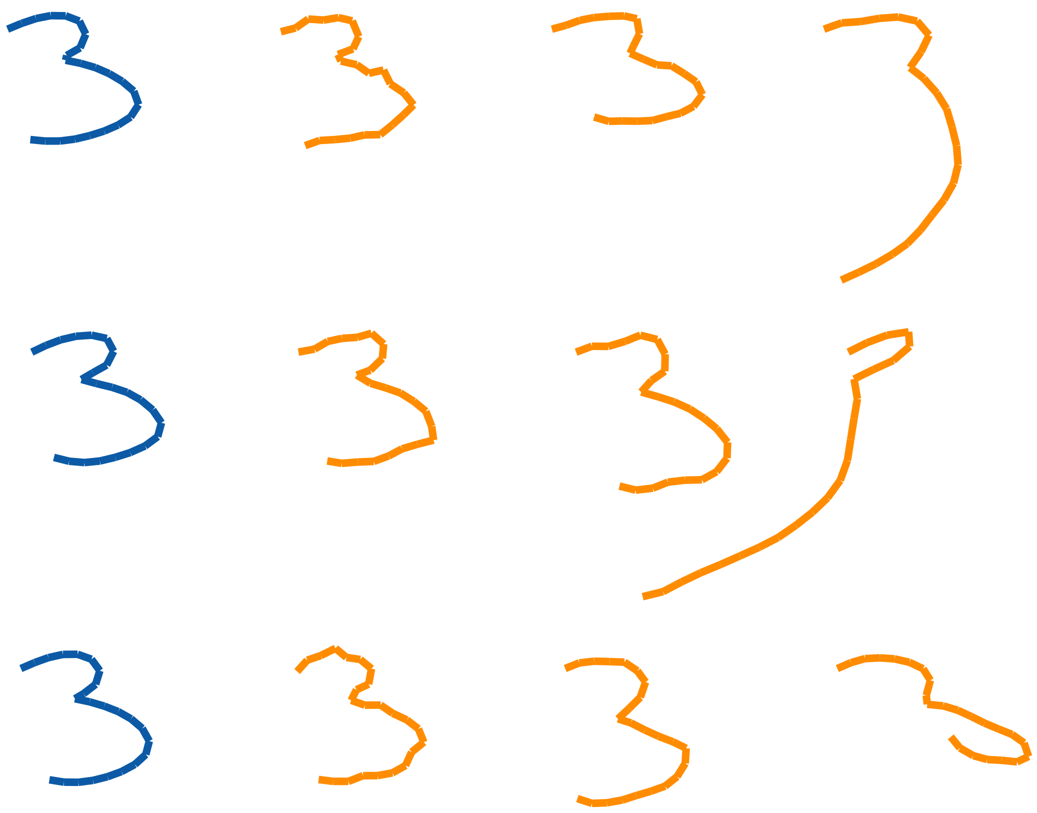


Figure 5.3: *Adding random noise to the points of a stroke (second column), its first derivative (third column) and second derivative (fourth column). Three different times.*

The examples chosen here for the results on the second derivative are among the most extreme one that can be found with Widget A.7. However, it is telling that the difference between noise on a stroke and its first derivative compared to the original is much smaller than between noise on the second derivative and the original.

A problem here to consider is that the effect of random dispositions gets amplified through the application of the discrete integrals. The end point of $\Omega_{n,\varepsilon}(s)$ is at most ε away from P_n. However, the end point of $\left(\int_{P_1}^{n-1} \circ \Omega_{n-1,\varepsilon} \circ d_n\right)(s)$ can be up to $(n-1) \cdot \varepsilon$ away from P_n. So, the effect of perturbing the k-th derivative of a stroke will be visually larger the higher k is. To judge the effect on the actual shape of strokes better, we normalise the resulting strokes such that they have the same start and end point as the original one. The results can be seen in Figure 5.4. We see that the visual difference to the original stroke is still much higher once the second derivative is affected; at least in these examples.

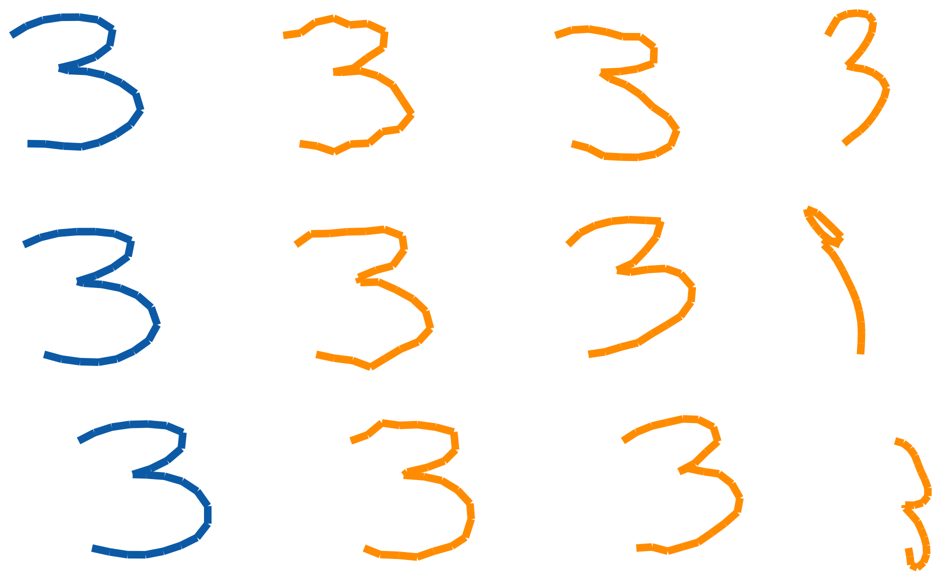


Figure 5.4: *Adding random noise to the points of a stroke (second column), its first derivative (third column) and second derivative (fourth column). Normalising the results to the same start and end point.*

None of this is a real surprise though. A back-of-the-envelope calculation shows that the distance the endpoint moves has a variance proportional to n if noise is added to the first derivative, and a variance proportional to n^3 if the noise is added to the second derivative. However, as we argued before we focus on comparing strokes with similar Length in practise. So, the sample rate n is a constant in our considerations. Hence, we will only focus on the jump in visual change to prompt the more structured observations in the next section.

5.2 Determinants

The above section serves as a motivation to why looking at the curvature of a stroke might be interesting. Here we want to formulate this idea in a more mathematically sound way. First, we want to consider a different term than the actual curvature that describes something similar.

We looked at the second discrete derivative of a stroke given by

$$\kappa_i = P_{i-1} - 2P_i + P_{i+1}.$$

The curvature itself is then $|\kappa_i|$ and it is equal to the length of the diagonal of P_i in the parallelogram spanned by P_{i-1}, P_i and P_{i+1}. Now, assume[3] that the neighbouring points along a stroke have constant distance d. Then, the three points P_{i-1}, P_i, P_{i+2} form an isosceles triangle. Its height in the point P_i is $h_i := \frac{1}{2}|\kappa_i|$ and, subsequently, its area is

$$A_i := h_i \cdot \sqrt{d^2 - h_i^2} = \frac{1}{4} \cdot \left(|\kappa_i| \cdot \sqrt{4d^2 - |\kappa_i|} \right).$$

As d is a constant here, the area is uniquely determined by the curvature of the stroke. But as we have seen in Section 2.1.1, the area of this triangle can also be computed via the determinant $A_i = \frac{1}{2}[\hat{P}_{i-1}, \hat{P}_i, \hat{P}_{i+1}]$.

We use this fact to use the determinant as a substitute for the curvature even if the points along the stroke do not have constant distance d. It is handy because of its algebraic and geometric properties. Moreover, it can be negative, so it even captures the sign of the curvature directly.

However, the mapping $|\kappa_i| \mapsto \frac{1}{2}[\hat{P}_{i-1}, \hat{P}_i, \hat{P}_{i+1}]$ we get is not one-to-one: the triangle area becomes zero both for $|\kappa_i| = 0$ and $2d$ (and maximal for $|\kappa_i| = \frac{1}{2}\sqrt{2}d$ when the triangle is equilateral). In order to obtain a reconstruction method similar to the discrete integration $\int_S^n$ we have to measure additional information. We will look at other determinants, namely $[\hat{P}_1, \hat{P}_i, \hat{P}_n]$. Whereas $[\hat{P}_{i-1}, \hat{P}_i, \hat{P}_{i+1}]$ describes the local curvature directly at the point P_i, this new de-

[3]As already mentioned, this is an assumption one has to make in the continuous case to even define the curvature properly.

terminant can be thought of a global-scale curvature comparing the position of P_i with the start and end point.

Below we first show how looking at both kinds of determinants at the same time allows us to reconstruct strokes. Afterwards, we show that they form a more natural moduli space of strokes than $\left(\mathbb{R}^2\right)^n$ in the sense that the geometric transformations from Chapter 3 operate on them in a very simple manner.

Definition 5.2.1: We call $l_i(s) := [\hat{P}_{i-1}, \hat{P}_i, \hat{P}_{i+1}]$ for $i = 2, ..., n-1$ a **local** determinant of s and $g_i(s) := [\hat{P}_1, \hat{P}_i, \hat{P}_n]$ for $i = 1, ..., n$ a **global** determinant of s.

Furthermore, define the **local shape vector** of s by

$$l(s) = (l_2(s), ..., l_{n-1}(s))^T \in \mathbb{R}^{n-2},$$

the **global shape vector** of s by

$$g(s) = (g_2(s), ..., g_{n-1}(s))^T \in \mathbb{R}^{n-2}$$

and the **(full) shape vector** of s by

$$\Delta(s) = (l_2(s), ..., l_{n-1}(s), g_2(s), ..., g_{n-1}(s))^T \in \mathbb{R}^{2(n-2)},$$

△

Note that
$$g_1(s) = [\hat{P}_1, \hat{P}_1, \hat{P}_n] = 0 = [\hat{P}_1, \hat{P}_n, \hat{P}_n] = g_n(s)$$

holds for all strokes s. We see them as part of the global shape data of s, but we do not actively track them in $g(s)$ or $\Delta(s)$. Also, if we normalise strokes to lie in $\mathfrak{Q}$, the values of both local and global determinants lies in the interval $[-1, 1]$ and so they are proto-feature. As they describe triangle areas, it is evident that they are shape proto-features.

Definition 5.2.2: We say a stroke s is in **general position** if no line $P_i \vee P_{i+1}$ is parallel to $P_1 \vee P_n$, for all $i = 1, ..., n-1$. △

Note that strokes which are in general position form a dense subset of $\mathfrak{S}$. In particular, we can approximate any recorded stroke arbitrarily well by a stroke in general position.

Definition 5.2.3: For a stroke s define the vector

$$b_i^s := \begin{pmatrix} l_i(s) \\ g_i(s) \\ 1 \end{pmatrix}$$

and the matrices A_i^s by the equation

$$A_i^s \hat{P}_i = b_i^s$$

for all $i = 3, ..., n$. The matrices are explicitly given by

$$A_i^s = \begin{pmatrix} \left(\hat{P}_{i-2} \times \hat{P}_{i-1}\right)^T \\ \left(\hat{P}_n \times \hat{P}_1\right)^T \\ 0 \;\; 0 \;\; 1 \end{pmatrix}.$$

△

Proof. The explicit formula follows from the equation $[X, Y, Z] = (X \times Y)^T Z$ for arbitrary vectors $X, Y, Z \in \mathbb{R}^3$. Using it we can see that the first two entries of $A_i^s \hat{P}_i$ are exactly the local and global determinant in b_i^s. □

Theorem 5.2.4: *Let $s = (P_i)_{i=1}^n$ be a stroke in general position and let $A, B, C \in \mathbb{R}^2$ be three points which are not collinear. Build a stroke $t = (Q_i)_{i=1}^n$ recursively via the following rule:*

1. Set

$$Q_1 = A \quad Q_2 = B, \quad Q_n = C.$$

2. Set

$$Q_i = \left(A_i^t\right)^{-1} b_i^s$$

for all $i = 3, ..., n - 1$.

Note that we use the matrix A_i^t, but the vector b_i^s. So, this definition constructs a new stroke t from A, B, C and the previously built points on t using the local and global determinants of s. Then the following are equivalent:

(1.) $[\hat{A}, \hat{B}, \hat{C}] = g_2(s)$.

(2.) $\Delta(s) = \Delta(t)$.

(3.) There exists an affine transformation M with determinant 1 *such that $s = Mt$.*

Proof. (3.) $\Rightarrow$ (2.): Matrices of determinant 1 do not change any determinant values.

(2.) $\Rightarrow$ (1.): If all local and global determinants are equal, then, in particular, the second global one is.

(1.) $\Rightarrow$ (3.): If an affine transformation exists that maps t to s it has to be the one that maps A, B, C to P_1, P_2, P_n, respectively. Assume, w.l.o.g., that P_1 is the origin, that P_n lies on the positive x-axis and that P_2 lies above the x-axis. Then there exits an Euclidean transformation that maps A to the origin and C to the positive x-axis. As

$$[\hat{A}, \hat{B}, \hat{C}] = g_2(s) \overset{\text{def}}{=} [\hat{P_1}, \hat{P_2}, \hat{P_n}],$$

the point B has to be in the same half-space with respect to $A \vee C$ as P_2 is with respect to $P_1 \vee P_n$. I.e., it also lies above the x-axis. Then we apply a scaling of the form

$$\begin{pmatrix} \frac{1}{d} & & \\ & d & \\ & & 1 \end{pmatrix}$$

for an appropriate $d > 0$ such that C lands on P_n. It leaves triangle areas invariant and thus the image of B must have the same y-coordinate as P_2. Finally, a shearing parallel to the x-axis maps B to P_2. All these three intermediate transformations have determinant 1, thus their concatenation M has so, too.

What is left to show is that building t after this realignment leads to $t = s$ and that building t from the original A, B, C and then applying M leads to the same result. The former can be done by induction: P_1, P_2 and P_n already coincide with Q_1, Q_2 and Q_n, respectively. So now assume that $P_1, ..., P_{i-1}$ and P_n coincide with $Q_1, ..., Q_{i-1}$ and Q_n, respectively. As s is in general position, the lines $P_1 \vee P_n$ and $P_{i-2} \vee P_{i-1}$ are not parallel which implies that A_i^s is invertible. Because s and t are already equal up to the $(i-1)$-st point, we know that

$A_i^s = A_i^t$. So,

$$Q_i = \left(A_i^t\right)^{-1} b_i^s = \left(A_i^s\right)^{-1} b_i^s = P_i.$$

We can reformulate this geometrically: Up to a factor of 2, the determinant $[X, Y, Z]$ is the signed area of the triangle X, Y, Z. If we now fix two of these points, say X, Y, as the base of the triangle, the determinant is proportional to the corresponding altitude of the triangle or, in other words, to the signed distance of Z to the line $X \vee Y$.

So, when we want a point Q_i with $[\hat{P}_{i-2}, \hat{P}_{i-1}, \hat{Q}_i] = [\hat{P}_{i-2}, \hat{P}_{i-1}, \hat{P}_i]$ and with $[\hat{P}_1, \hat{Q}_i, \hat{P}_n] = [\hat{P}_1, \hat{P}_i, \hat{P}_n]$, it has to have the specific distance given by

$$\frac{[\hat{P}_{i-2}, \hat{P}_{i-1}, \hat{P}_i]}{2\,\|P_{i-2} - P_{i-1}\|_2}$$

to the line $P_{i-2} \vee P_{i-1}$ and the distance

$$\frac{[\hat{P}_1, \hat{P}_i, \hat{P}_n]}{2\,\|P_n - P_1\|_2}$$

to $P_1 \vee P_n$. This point is unique as long as $P_1 \vee P_n$ and $P_{i-2} \vee P_{i-1}$ are not parallel and P_i is one such point; so $Q_i = P_i$.

This last argument shows that this construction is compatible with the affine transformation M: It leaves determinants and, hence, triangle areas invariant. Tt also does not change the (non-)parallelity of lines, since it is an affine transformation. That means the points Q_i we construct in the original setting have to coincide with $M^{-1}P_i$ for all i. □

Corollary 5.2.5: *Let s and t be two strokes with the same start and end points S and E. If $\Delta(t) = d \cdot \Delta(s)$ for some $d \in \mathbb{R}^\times$, then s and t differ by a shearing parallel to $S \vee E$ and a scaling perpendicular to $S \vee E$ by the factor d.*

Proof. W.l.o.g., assume that S is the origin and that E is on the positive x-axis. Shearing parallel to the x-axis then does not change any determinants, so we can also assume that the point P_2 on s and Q_2 on t have the same x-coordinate.

What is left to show is that $D\hat{s} = \hat{t}$ for

$$D = \begin{pmatrix} 1 & & \\ & d & \\ & & 1 \end{pmatrix}.$$

We know that $\Delta(D\hat{s}) = d \cdot \Delta(s)$, since scaling in one direction by d scales (triangle) areas by the same amount. Furthermore, $D(P_2)$ and Q_2 have to be equal. Then, Theorem 5.2.4 implies that $Ds = t$. □

The reconstruction process from Theorem 5.2.4 can be tested in Widget A.8. In particular, The stroke t can be built regardless of whether the points A, B and C are in the correct spatial relation. But the last point $\left(A_n^t\right)^{-1} b_n^s$ we can construct with the given rule coincides with C if and only if the triangles A, B, C and $P_1P_2P_n$ have the same area. Moreover, moving one of the points A, B, C parallel to the line through the other two does not change the triangle area which visualises Corollary 5.2.5.

These two statements tell us that if we keep track of basic position and orientation information of a stroke — the first, second and last point on it — we can reconstruct it from the shape vector up to shearing parallel to the start-end line. Seeing strokes that only differ by such a shearing as equivalent is reasonable as this is a common alteration in typography and chirography. However, as we argued in Section 3.1.1, strokes that emerge from arbitrary projective transformation should have the same form as the original stroke, too. In that section, we introduced the notion of compliant projective transformations to deal with certain rendering and display problems. For the theoretical considerations here we will ignore these concerns and assume that we can freely apply any projective transformation to strokes. The determinant multiplication rule implies that $\Delta(M(s)) = \det(M) \cdot \Delta(s)$ for a projective transformation M. And since we want to see s and $M(s)$ as equivalent, it suggests itself identifying scalar multiples of shape vectors.

Let $\mathfrak{S}_\sim$ be the set of all strokes $s \in \mathfrak{S}$ that are *not* straight line segments. I.e., those strokes s for which not all points $P_1, ..., P_n$ on it are collinear. Consider the function

$$\Gamma : \mathfrak{S}_\sim \to \mathbb{RP}^{2n-5}, \qquad s \mapsto [\Delta(s)].$$

Practically, it is indistinguishable from Δ. But we want to emphasise that we now consider shape vectors only up to scalar multiples.

Note that since the zero vector does not represent any object in projective space, we have to exclude all strokes s from the domain of Γ with $\Delta(s)$. And these are precisely the straight line segments. However, as we have seen, these particular strokes can very easily be recognised by their Straightness and classified solely by their orientation. So, it is no problem to exclude them here.

Theorem 5.2.6: *Let T be either an affine pull C_λ with $\lambda \in \mathbb{R}^\times$ or an acceleration $A_{\Delta v}$ by $\Delta v \in \mathbb{R}$. (See Sections 3.1.2 and 3.1.3.) It operates on* $\operatorname{im}(\Gamma)$ *by*

$$T(\Gamma(s)) := \Gamma(T(s))$$

and this action is given by a projective transformation on $\mathbb{RP}^{2n-5}$*. I.e., it operates linearly on the elements of* $\mathbb{RP}^{2n-5}$*.*

Proof. The functions in question form groups as shown in Lemmata 3.1.2 and 3.1.5. So, it is clear that the operation defined above forms a proper group action. To show the second part of the statement, we compute local and global determinants of transformed strokes and will see that the results are linear combinations of determinants of the initial stroke. Thus, we can write the action on $\mathbb{RP}^{2n-5}$ by matrix multiplication. That this matrix is invertible then follows simply from the fact that the functions we work with here are invertible.

To make the subsequent computations easier, we restrict ourselves to nailed strokes $\mathfrak{N} \cap \mathfrak{S}_\sim$. In particular, this means $g_i(s) = -y_i$. We can do this, since Euclidean transformations operate trivially on $\operatorname{im}(\Delta)$. Also, recall that we denote the i-th point of u by u_i.

Affine pulls: Let $\lambda \in \mathbb{R}^\times$ and $s \in \mathfrak{N} \cap \mathfrak{S}_\sim$. In this case, u is explicitly given by

$$\left(\left(\frac{i-1}{n-1}, 0\right)\right)_{i=1}^{n}.$$

With this, an the fact that s and u have the same start and end points, we directly get the global determinant

$$\begin{aligned} g_i(C_\lambda(s)) &= [\underbrace{\lambda\hat{P}_1 + (1-\lambda)\hat{u}_1}_{=\hat{P}_1=\hat{u}_1},\ \lambda\hat{P}_i + (1-\lambda)\hat{u}_i,\ \underbrace{\lambda\hat{P}_n + (1-\lambda)\hat{u}_n}_{=\hat{P}_n=\hat{u}_n}] \\ &= \lambda\cdot[\hat{P}_1,\hat{P}_i,\hat{P}_n] + (1-\lambda)\cdot\underbrace{[\hat{u}_1,\hat{u}_i,\hat{u}_n]}_{=0} = \lambda g_i(s) \end{aligned}$$

for every $i = 1, ..., n$. The local determinant $l_i(C_\lambda(s))$, for $i = 2, ..., n-1$, we can transform to

$$\begin{aligned} l_i(C_\lambda(s)) =&[\lambda\hat{P}_{i-1} + (1-\lambda)\hat{u}_{i-1},\ \lambda\hat{P}_i + (1-\lambda)\hat{u}_i,\ \lambda\hat{P}_{i+1} + (1-\lambda)\hat{u}_{i+1}] \\ =&\lambda^3[\hat{P}_{i-1},\hat{P}_i,\hat{P}_{i+1}] \\ &+ \lambda^2(1-\lambda)\left([\hat{P}_{i-1},\hat{P}_i,\hat{u}_{i+1}] + [\hat{P}_{i-1},\hat{u}_i,\hat{P}_{i+1}] + [\hat{u}_{i-1},\hat{P}_i,\hat{P}_{i+1}]\right) \quad \text{(a)} \\ &+ \lambda(1-\lambda)^2\left([\hat{u}_{i-1},\hat{u}_i,\hat{P}_{i+1}] + [\hat{u}_{i-1},\hat{P}_i,\hat{u}_{i+1}] + [\hat{P}_{i-1},\hat{u}_i,\hat{u}_{i+1}]\right) \quad \text{(b)} \\ &+ (1-\lambda)^3[\hat{u}_{i-1},\hat{u}_i,\hat{u}_{i+1}] \end{aligned}$$

using the multi-linearity of the determinant. Furthermore, we compute

$$[\hat{P}_{i-1},\hat{u}_i,\hat{P}_{i+1}] = \begin{vmatrix} x_{i-1} & \frac{i-1}{n-1} & x_{i+1} \\ y_{i-1} & 0 & y_{i+1} \\ 1 & 1 & 1 \end{vmatrix} = -[P_{i-1},P_{i+1}] + \frac{i-1}{n-1}(y_{i+1}-y_{i-1})$$

and similarly

$$[\hat{P}_{i-1},\hat{P}_i,\hat{u}_{i+1}] = [P_{i-1},P_i] - \frac{i}{n-1}(y_i - y_{i-1})$$

and

$$[\hat{u}_{i-1},\hat{P}_i,\hat{P}_{i+1}] = [P_i,P_{i+1}] - \frac{i-2}{n-1}(y_{i+1}-y_i).$$

Note that on the right-hand sides we have 2×2-matrices. With this we can write the right-hand factor in term (a) above as

$$\begin{aligned} &[P_{i-1},P_i] - [P_{i-1},P_{i+1}] + [P_i,P_{i+1}] \\ &-\frac{i}{n-1}(y_i - y_{i-1}) + \frac{i-1}{n-1}(y_{i+1}-y_{i-1}) - \frac{i-2}{n-1}(y_{i+1}-y_i). \end{aligned}$$

The first three summands here are, after a sign flip in the second one, twice the area of the triangle $P_{i-1}P_iP_{i+1}$, which is equal to $l_i(s)$. And the second three summands can simply be added up. Moreover, we replace the y-coordinates by negative global determinants. The last factor in (a) is then

$$l_i(s) - \frac{1}{n-1}\left(g_{i-1}(s) - 2g_i(s) + g_{i+1}(s)\right). \tag{a'}$$

Next, we compute

$$[\hat{u}_{i-1}, \hat{P}_i, \hat{u}_{i+1}] = \begin{vmatrix} \frac{i-2}{n-1} & x_i & \frac{i}{n-1} \\ 0 & y_i & 0 \\ 1 & 1 & 1 \end{vmatrix} = -\frac{2}{n-1} y_i$$

and similarly

$$[\hat{u}_{i-1}, \hat{u}_i, \hat{P}_{i+1}] = \frac{1}{n-1} y_{i+1}$$

and

$$[\hat{P}_{i-1}, \hat{u}_i, \hat{u}_{i+1}] = \frac{1}{n-1} y_{i-1}.$$

This allows us to re-write the last factor in term (b) to

$$-\frac{1}{n-1}\left(g_{i-1}(s) - 2g_i(s) + g_{i+1}(s)\right). \tag{b'}$$

When we now replace the right-hand factors in (a) and (b) by (a′) and (b′) in our initial computations of $l_i(C_\lambda(s))$ we get

$$\begin{aligned} l_i(C_\lambda(s)) =& \lambda^3 \underbrace{[\hat{P}_{i-1}, \hat{P}_i, \hat{P}_{i+1}]}_{=l_i(s)} \\ &+ \lambda^2(1-\lambda)\left(l_i(s) - \frac{1}{n-1}\left(g_{i-1}(s) - 2g_i(s) + g_{i+1}(s)\right)\right) \\ &- \lambda(1-\lambda)^2\left(\frac{1}{n-1}\left(g_{i-1}(s) - 2g_i(s) + g_{i+1}(s)\right)\right) \\ &+ (1-\lambda)^3 \cdot \underbrace{[\hat{u}_{i-1}, \hat{u}_i, \hat{u}_{i+1}]}_{=0} \\ =& \lambda^2 \cdot l_i(s) - \lambda(1-\lambda) \cdot \frac{1}{n-1} \cdot \left(g_{i-1}(s) - 2g_i(s) + g_{i+1}(s)\right). \end{aligned}$$

Acceleration: Let $\Delta v \in \mathbb{R}$ and abbreviate the normalising factor in the definition of accelerations (see Definition 3.1.3) by

$$N = 1 + (n-1) \cdot \Delta v.$$

First observe that global determinants stay invariant under $A_{\Delta v}$, since accelerations only affect x-coordinates of strokes (in $\mathfrak{N}$) and leave the y-coordinates invariant. And for local determinants we get

$$\begin{aligned} l_i(A_{\Delta v} s) &= \begin{vmatrix} \frac{1}{N}\left(x_{i-1} + \Delta v(i-2)\right) & \frac{1}{N}\left(x_i + \Delta v(i-1)\right) & \frac{1}{N}\left(x_{i+1} + \Delta v i\right) \\ y_{i-1} & y_i & y_{i+1} \\ 1 & 1 & 1 \end{vmatrix} \\ &= \frac{1}{N}\left(\begin{vmatrix} x_{i-1} & x_i & x_{i+1} \\ y_{i-1} & y_i & y_{i+1} \\ 1 & 1 & 1 \end{vmatrix} + \Delta v \cdot \begin{vmatrix} i-2 & i-1 & i \\ y_{i-1} & y_i & y_{i+1} \\ 1 & 1 & 1 \end{vmatrix} \right) \\ &= \frac{1}{N}\left(l_i(s) + \Delta v\left(y_{i-1} - 2y_i + y_{i+1}\right)\right) \\ &= \frac{1}{N}\left(l_i(s) - \Delta v\left(g_{i-1}(s) - 2g_i(s) + g_{i+1}(s)\right)\right). \end{aligned}$$

□

The above proof did not only demonstrate that affine pulls and accelerations operate linearly on determinants. It also shows that the local determinants get altered by a multiple of $g_{i-1}(s) - 2g_i(s) + g_{i+1}(s)$ in both cases. It is the negative of the y-coordinate of the second derivative κ_i of s at point P_i. That means that affine pulls and accelerations change one kind of curvature measure — local determinants — by a different kind of curvature measure.

This last theorem illustrates that the two ideas of this chapter — considering the curvature of strokes at all and representing it via determinants — and the geometric transformations from Chapter 3 complement one another. In particular, projective transformations and convex combinations were motivated mostly by graphical/visual considerations and accelerations by kinematic ones. Now, however, we see that they affect the shape of stroke in a simple and controlled way.

When we take this as reason enough to believe that $\Gamma(\mathfrak{S}_\sim) \subset \mathbb{R}\mathrm{P}^{2n-5}$ is a good moduli space for non-straight-strokes, the next step would be to analyse and classify sample sets in this space. In our pursuit to find classifiers based on concrete geometric properties, it is now a straightforward idea to use multi-homogeneous bracket polynomials. They are polynomials built from determinants of points such that every point appears the same number of time in every monomial. That makes these polynomials, in particular, invariant under projective transformations: if it is 0 for a specific point set, it is also zero for the transformed point set.

In the context of strokes this means the following: If we have a set of good samples for a stroke type and look at the images under Γ we can try to find an algebraic variety in $\mathbb{R}\mathrm{P}^{2n-5}$ that is given by multi-homogeneous bracket polynomials that interpolates the sample set (as good as possible). Then, there exists a concrete geometric property that describes the sample set (as good as possible). The problem herein is that the translation from bracket polynomials back to explicit geometric statements is tricky and non-obvious. See, for example, Neil White's work [68] on Cayley factorisation.

It would go beyond the scope of this thesis to discuss this idea. Instead, we want to close by giving three short examples of how polynomials in local and global determinants relate to geometric properties of strokes.

Example 5.2.7: Let $s \in \mathfrak{S}_\sim$ be a non-straight stroke. Then, the geometric property that

the first quarter part of s is straight

is equivalent to s being a solution to the following linear system on determinants[4]:

$$l_2(s) = l_3(s) = \dots = l_{\frac{n}{4}}(s) = 0.$$

This property is present in strokes that represent `7`, `L` or $\square$. Next, we make the

[4]We took the liberty to simply write the index $\frac{n}{4}$. Depending on n we would use $\lfloor \frac{n}{4} \rfloor$ or $\lfloor \frac{n}{4} \rfloor - 1$ to get the proper first fourth of points. Similar simplifications are applied throughout the example.

equations slightly more complicated. The linear system

$$l_2(s) - l_3(s) = l_3(s) - l_4(s) = \ldots = l_{\frac{n}{2}-1}(s) - l_{\frac{n}{2}}(s)$$

encodes that the curvature is constant for the first half of the stroke. So, it describes the property that

the first half of the stroke forms a circular arc.

Note that this tells us nothing about the angle of the arc. So, this describes strokes representing `S` as well as `9`.

Next, we can look at the single polynomial

$$\sum_{i=2}^{n-1} l_i(s).$$

When the curvature value at every point average out — i.e., when this polynomial evaluates to zero — we can say that

the stroke has to have as many (local) left turns as right turns.

This describes, for examples, strokes types like `2`, `S` and `8`; but also straight lines for which every summand is zero. Moreover, the zero set of this polynomial separates the space $\mathbb{RP}^{2n-5}$ into two parts. The larger the absolute value of $\sum_{i=2}^{n-1} l_i(s)$ for a concrete stroke s, the more lopsided the curvature distribution is — at least when we only look at strokes $s \in \mathfrak{Q} \cap \mathfrak{S}_{\sim}$ such that all summands are bounded. However, that means that not only points $\Gamma(s)$ on the surface given by the polynomial can represent stroke types, but also points in each half-space. E.g., good samples of `7` and `L` do not lie on the hyper-surface but are separated by it. △

The example above illustrates how determinants in local and global determinants can model geometric properties. In particular, the last polynomial is a continuous variant of the features Left and Right Curvature we defined in Section 2.3.3 via counting functions. Moreover, we saw in the introduction of the Kolmogorov-Smirnov test a variant of the Straightness of a stroke given by $\sum_{i=2}^{n-1} g_i(s)$; even though we described it there a bit differently.

So, not only do vectors with local and global determinants as their entries serve as an elegant way to describe the shape of a stroke that is stable under transformations. Also, polynomials in these determinants have the potential to describe specific geometric properties of this shape.

Looking back at the primary source of ALICE:HWR, the article [11] by Delaye and Anquetil, we can see that some of the features listed there have very convoluted definitions; especially the Proportion of downstrokes trajectory. So, searching for (multi-homogeneous) polynomials in determinants may serve as a way to find simple features that still convey a lot of geometric meaning of the original strokes.

6 Looking back at ALICE:HWR

"Where is the library?"
"Turn right, proceed thirty-four paces, turn right again, twelve paces, then through door on the right, thirty-five paces, through archway on right another eleven paces, turn right one last time, fifteen paces, enter the door on the right."
Mappo stared at Iskaral Pust. The High Priest shifted nervously.
"Or," the Trell said, eyes narrowed, "turn left, nineteen paces."
"Aye," Iskaral muttered.

— *Deadhouse Gates*, by Steven Erikson

Some of the ideas presented in the last three chapters can be implemented directly. For example, measured values often have to be bounded to be useful. So, normalising them to get (proto-)feature suggests itself. Here in this chapter we talk about some of the nuances and describe how the ideas from the last chapters found their way into ALICE:HWR.

To start, we state the whole training and classification process of ALICE:HWR — similar to how we did it in Section 2.3.1. Afterwards we will talk about certain points in more detail. We will describe version 4 of ALICE:HWR which is the one found currently (as of January 2019) in the ALICE iBook and which was used in the second study, at *Mittelschulen* in 2017. We will compare it qualitatively to version 3 which was used in the first study, at *Gymnasien* in 2016. Version 4 can be found in the companion iBook [67] as Widget A.9.

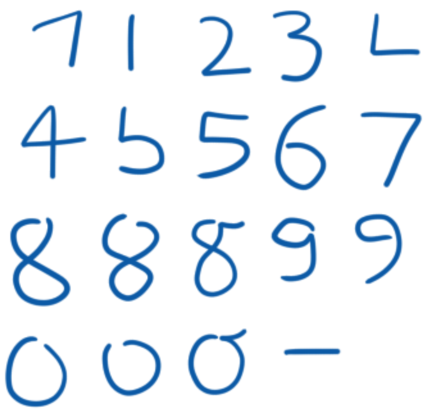


Figure 6.1: *Stroke types used in ALICE:HWR*

Recall the alphabet used in ALICE:HWR

$$\begin{aligned}\mathbb{A} = \{&\texttt{1a, 1b, 2, 3, 4a,}\\ &\texttt{4b, 5a, 5b, 6, 7,}\\ &\texttt{8a, 8ar, 8b, 9a, 9b,}\\ &\texttt{0a, 0ar, 0b, -}\}.\end{aligned}$$

and its "ideal" realisations seen in Figure 6.1 we introduced already in Section 2.3.1. In Sections 1.2.2 and 1.3.2 we argued that we want to recognise numbers based on how these strokes were written — i.e., how the finger/pen of the user moved along the touchscreen — and the image that was produced. In particular, we assume that every stroke is written in the "proper" way taught in elementary school.

For practical reasons we enabled the possibility to write any stroke backwards. It is implemented by changing the order of a recorded stroke and then classifying the original and the reversed stroke separately. Whichever gives the better result is viewed as the intended user input. This is necessary for horizontal lines (used in `5`'s and (German) `7`'s) as left-handed and right-handed person write them with different orientation. For all other strokes this is mostly a luxury feature. We will not list this as an essential part of the classification process below. Note, however, that for the closed-loop strokes that represent a `0` or `8` we encoded the reversely written versions as separate stroke types. This produces more reliable results.

In Sections 1.2.2 and 1.3.2 we also mentioned that the HWR algorithm should be as simple as possible in the sense that we want a small set of training data and an uncomplicated classification process. Below we will discuss how this was achieved based on the theoretical consideration of the previous chapters.

6.1 Training

In Section 2.3.1 we already sketched the general idea of the classifier: we want concrete rules for when a stroke type is definitely *not* a match for a recorded stroke, and a fuzzy distance function that thereupon ranks the remaining types based on idealised feature vectors. The training process to find these rules and feature vectors looks like this:

1. Five sample strokes were recorded for every type in $\mathbb{A}$ and preprocessed by Algorithm 2.3.4.
2. The 25 different features from Section 2.3.3 were computed for all these strokes.
3. They were scaled with the function $\rho_{3,0.7}$ (defined in Section 2.3.3) to make values larger than 0.7 even larger; and smaller values smaller.
4. We built the training context $\mathcal{T}$ and scaled it in two different ways. First, such that low feature values always imply high values. And the other time, such that high values imply low ones.
5. The program *Concept Explorer* (see [71]) was used to find exclusion rules for each stroke type and for each scaling of $\mathcal{T}$. All decision areas that exclude a stroke type were joined to form a single, large decision area.
6. The sample strokes were multiplied using the Gaussian filter $\mathcal{G}_{A_{0.02}}$ and the projective transformation M from Example 3.2.2. Similar as described in the example, they are applied 9 times each, based on a Kolmogorov-Smirnov test to determine how much the feature values change.
7. The feature vectors for all transformed and untransformed strokes are computed and the average of all feature vectors associated to a stroke type is computed. This serves as an idealised feature vector for each stroke type.

Some noteworthy facts here are the following: First, the number of samples used to train the algorithm is very small. Five per stroke type, so, 95 in total. They were all produced by a single person who is left-handed.

Second, features were modulated with $\rho_{3,0.7}$, as explained in Section 2.3.3, since they describe very concrete geometric properties. And perfect representatives of stroke types are expected to either have them or not have them with 100% certainty. This is not true for all features, but for many. Features for which values around $\frac{1}{2}$ have significance[1] are usually also represented by their complement. (E.g., LEFT and RIGHT CURVATURE.) So, the bias introduced by $\rho_{3,0.7}$ is balanced out.

Third, out of the $2^{25} \approx 3.6 \cdot 10^7$ possibilities to scale the training context, only two were used. Moreover, only a small number of exclusion rules was chosen.

The basic idea how these rules can be found was stated in Section 4.2. But for ALICE:HWR we did not use hypotheses as explained there, but generatives. See [14] for a definition. Generatives are subsets of hypotheses and give, consequently, larger decision rule areas. So, they are more lenient in deciding when a stroke type has to be ruled out.

Additionally to this generalisation, the decision rules were made even larger by heuristic choices based on the idea of attribute exploration: The process by which an implication basis for finite contexts can be found is based on an order of the premises. Furthermore, the implications are computed following this order. Then, a human user can supervise this process and interrupt it at any point. If they disagree with a particular implication, they can provide a counterexample that will be included in the context and taken into account in the computations of the following implications. And the order of the premisses just mentioned guarantees that the implications found up to this point are still valid.

In the training of ALICE:HWR this veto process was executed by manually changing the found decision rules. Adding concrete counterexamples would have prolonged the run time of the attribute exploration via *Concept Explorer*.

[1]i.e., features that are better implemented as proto-features

Concretely, if an exclusion rule indicated that a certain feature f_i should have a value larger than 0.9 but a counter-example was known, this condition in the exclusion rule ways simply dropped. Similarly, all conditions demanding that features f_j should be larger than 0.1 were left out, too: First, this condition holds for many strokes anyway, because 0.1 is quite small. Second, because of the small sample size used, this bound becomes even more unreliable.

The equivalent steps were made for the opposite scaling of the training context and the resulting decision rules. In the end, there were between two and eight conditions left for different features in each exclusion rule.

Fourth, multiplying the sample strokes via geometric transformations, as explained in Example 3.2.2, was done to factor in more variations when computing the idealised feature vectors of each stroke type. The Kolmogorov-Smirnov test was not used as a factual assessment of the behaviour of the measured features, but as a tool for exploration. In particular, the number of times we can apply the transformations in question was found by the following heuristic:

It is the smallest number k for which 10% of all feature values of a stroke type change. I.e., applying the transformation T in question to all samples $\mathfrak{T}_l$ of a type $l \in \mathbb{A}$ multiple times leads to $f_i(\mathfrak{T}_l)$ and $f_i\left(T^k(\mathfrak{T}_l)\right)$ being differently distributed according to the Kolmogorov-Smirnov test for some features f_i. The value $k = 9$ is the smallest such that this happens for three of the 25 features. This was deemed a reasonable threshold to say that the transformed samples do not exactly depict the same thing as the original ones. That the value is the same for both the Gaussian filter and the acceleration is coincidental.

Fifth, in previous iterations of ALICE:HWR, the exclusion rules were heuristically determined. The method using Formal Concept Analysis shown here is, in contrast, more automatic.

6.2 Classification

The actual classification process is as described in Section 2.3.1. First, stroke types were excluded via rules found with Formal Concept Analysis. Second, the remaining types were ranked to find the best match using a function

$$\Phi : [0,1]^m \times [0,1]^m \to [0,1].$$

Concretely:

Algorithm 6.2.1: Stroke classification by ALICE:HWR

```
Input  : An alphabet $\mathbb{A}$, an exclusion rule $E_l$ for each $l \in \mathbb{A}$, a fuzzy
         vector $F_l \in [0,1]^m$ for each $l \in \mathbb{A}$ and a stroke $r \in \bigcup_{d=1}^{\infty} \left(\mathbb{R}^2\right)^d$.
Output: A stroke type $l \in \mathbb{A}$ and a fuzzy value $v \in [0,1]$ describing it
         likelihood to match $r$.
1 Pre-process $r$ via Algorithm 2.3.4 and replace $r$ by the result.
2 Using the 25 features from Section 2.3.3, compute the feature vector
   $F \leftarrow (\rho_{3,0.7}(f_1(r)), \ldots, \rho_{3,0.7}(f_{25}(r)))$.
3 Initiate $c \leftarrow \mathbb{A}$.
4 for $l \in \mathbb{A}$ do
5     if $F \in E_l$ then
6         $c \leftarrow c \setminus \{l\}$.
7 Sort $c$ by $l \leq l'$ iff $\Phi(F(r), F_l) \leq \Phi(F(r), F_{l'})$.
8 Return the last entry $l$ in $c$ and the fuzzy value $v = \Phi(r, v_l)$.
```

In version 3 of ALICE:HWR, which was used in the 2016 study, the comparison function Φ was basically the multi-matching function Σ from Section 4.3. I.e., the entries of a vector in $[0,1]^m$ were seen as the shifted feature values of proto-features.[2] So the entries were mapped into the interval $[-1,1]$ via the map $x \mapsto 2x - 1$. Then, the un-shifted vector $F(r), F_l \in [-1,1]^m$ were sorted by the values $\Sigma(F(r), F_l) = F(r)^T F_l$.

In constrast to the general procedure presented in Section 4.3, however, the summands in the scalar product were weighted: a diagonal $n \times n$-matrix W

[2]As seen in Section 2.3.3, many features used in ALICE:HWR have natural definition and interpretation as proto-features.

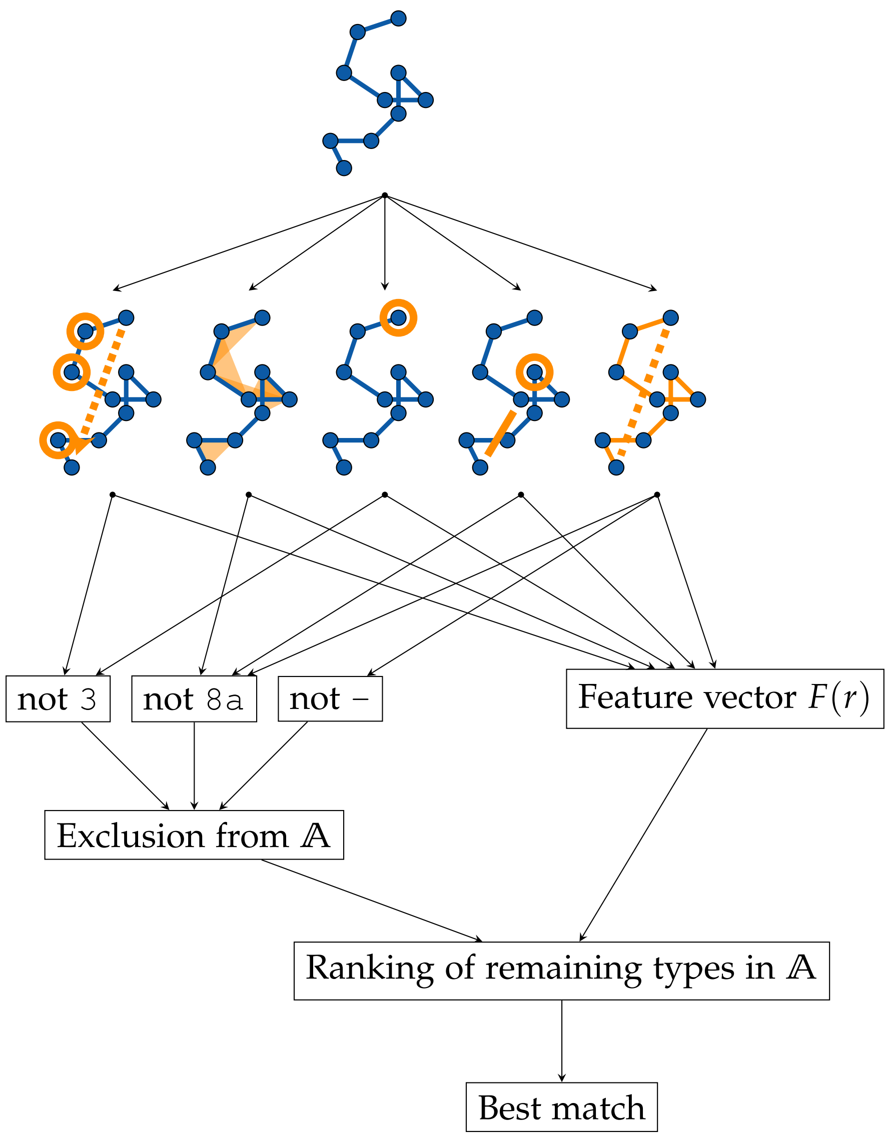

not 3
not 8a
not –
Feature vector $F(r)$
Exclusion from $\mathbb{A}$
Ranking of remaining types in $\mathbb{A}$
Best match

Figure 6.2: *Classification in ALICE:HWR.*

was built such that it has non-negative diagonal elements and trace 1. Then, the modified function $\Sigma'(u, v) := u^T W v$ was used.[3]

This is an artefact of using all 25 features for finding exclusion rules as well as this ranking step. Individual features, which were included to separate certain stroke types in the exclusion step, might not be beneficial in the ranking step together with all other features.

Another characteristic of version 3 is that it used the best two matches l, l'

[3]This description of Σ' lends itself directly for generalisation via other matrices W. Then, however, the interpretations from Chapter 4 are not directly applicable anymore.

with respect to Φ. Then, these two candidates were analysed using a features specific for this pair; but only if the fuzzy values $\Phi(F(r), F_l)$ and $\Phi(F(r), F_{l'})$ were close to each other.

In version 4 of ALICE:HWR, the function Φ was a variation of Π from Section 4.3. So, it was defined as

$$\Phi(u, v) = \frac{u^T v}{\|u\|_2 \, \|v\|_2}.$$

The difference to Π is that Φ is defined on $[0, 1]^m$ instead of $[-1, 1]^m$. This was done as it gave better results during testing; in certain border cases when strokes were written sloppy. As in the case with Σ above, this can be (at least partially) explained by the fact that not all features were created equal. In particular, many entries of the idealised proto-fuzzy vectors, representing the stroke types, were not close to the end of the interval $[-1, 1]$. This mostly occurred at entries of proto-features which were not relevant for that particular type.

Finally, note that Algorithm 6.2.1 always returns a stroke type l as the best match, independent of how high $\Phi(F(r), F_l)$ is. So, ALICE:HWR will always produce a number and never say *unrecognisable*.

6.3 Parsing

The last part of the classification process in ALICE:HWR is parsing: Finding out which digits and numbers were written, when several strokes were used. In ALICE:HWR this was implemented in a very simple way: Using the stroke types as given at the beginning of this chapter, there are only three digits that are composed of several strokes. They are

— The digit `4` built from `4a` and `1b`.

— The digit `5` built from `5a` and `-`.

— The digit `7` built from `7` and `-`.

There are two things to notice here: First, all these compound symbols consist of exactly two strokes. Second, there is at least one stroke in each digit that does not represent a digit alone. Concretely, these are `4a`, `5a` and `-`. So, having a list of recorded strokes $(r_1, ..., r_R)$ with $R \geq 2$, it is very easy to parse them into a number:

1. For all $i = 1, ..., R$ find a stroke type l_i for r_i that matches it best; using Algorithm 6.2.1.
2. If $l_i \in \{\texttt{4a}, \texttt{5a}, \texttt{-}\}$, find the closest stroke r_j next to r_i such that there is no other stroke in between and such that they form either a `4`, `5` or `7`. Label the set $\{r_i, r_j\}$ with the appropriate digit.
3. Parse the remaining strokes into the digit they represent (alone).

When we search for the closest strokes in step 2, we simply compute the center of mass of each stroke and then use its x-coordinate. If all digits are written horizontally and with a small gap between them, this algorithm will always work. This is due to the aforementioned properties of the compound digits.

Note that the algorithm would parse the strokes shown in Figure 6.3 to the number `45`. The reason is, of course, that we use only the horizontal distance of the strokes and only use their relative distance. So, the fact that `1b` is too far left and that the `-` is too far below, is ignored. Parsing strokes in this is justified, however, as we assume that the user actually wants to write a number and also wants it to be recognised.[4]

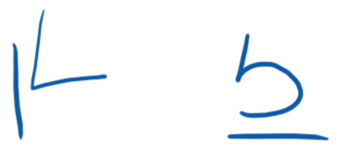


Figure 6.3: *A collection of strokes that will be recognised as the number* `45`.

Moreover, ALICE:HWR uses some additional ways to build these digits: For examples, `1a` and `-` will also be parsed as a `7`, and `5a` and `-` will be parsed as a `5`. The first is reasonable as stroke representing `1a` and `7` are basically indistinguishable. The second is handy in practice, because children often add additional strokes to correct their written characters.[5] So, they might write a stroke that is recognised as a `5a`; but for them it looks like a `5b`. So, they add a `-` to fix it. For a complete list of such additional building rules see Appendix B.

As explained in the last section, version 3 of ALICE:HWR uses the best two matches found in Algorithm 6.2.1. Both of them were incorporated during parsing: When a stroke type l was only the second best but allowed for one of the possible combinations, it was usually preferred over the best match.

[4]There are, of course, almost arbitrarily many ways to create false positives in any pattern recognition process.

[5]See Chapter 1 and [44].

6.4 Performance

The versions of ALICE:HWR used in the studies conducted in 2016 and 2017, did not directly test their solution rate. There are two reasons for that: First, the studies were intended to test the explanations and exercises about fractions. So, adding feedback questions whether the written numbers were recognised would have distracted from the main goal. Second, saving the written strokes for a later analysis was impossible since the memory in the iBook was limited and needed for other things.

There are, however, two assessments of how good ALICE:HWR performs: Interviews with the teachers after each study and an informal test with employees of the TUM.

The feedback to ALICE:HWR by teachers and pupils who participated in the ALICE studies was positive overall. Some children complained that it does not work. In all (reported) cases except one, this was fixed after the teacher told them to write the numbers properly.[6]

The test with TUM employees gave the following results: 11 employees of the Department of Mathematics and the School of Education were asked to write numbers onto an iPad. The numbers were randomly selected as integers in the interval $[1, 99]$. The interface to do so was identical to Widget A.9. Moreover, they were requested to write both nicely and quickly at the same time. When they were dissatisfied with their input, they were free to delete it before pressing the record button. This decision was left to the participants themselves. Moreover, they were stopped after approximately five minutes of writing, independent of how much they have written. The individual solution rates of the participants are:

$$\frac{14}{16}, \quad \frac{45}{45}, \quad \frac{41}{42}, \quad \frac{51}{55}, \quad \frac{37}{42}, \quad \frac{36}{40}, \quad \frac{47}{47}, \quad \frac{62}{65}, \quad \frac{38}{49}, \quad \frac{31}{35}, \quad \frac{90}{95}$$

The minimum is $\frac{38}{49} \approx 77.6\%$ while the maximum is 100%. And the total solu-

[6]I.e., how it was taught in elementary school.

tion rate is

$$\frac{492}{531} \approx 92.7\%.$$

The results were much higher when the *Apple Pencil* was used. In particular, hook removal was almost never necessary when this special stylus was used. But the algorithm was built for the user with a finger, so we did not test the solution rates for the Apple Pencil explicitly.

This solution rate is still far from sufficient. But considering the essential parts of the training and classification process, it is already satisfactory. It illustrates that the general design choices made for ALICE:HWR were reasonable — in particular, the combination of exclusion rules and and a fuzzy ranking step. Moreover, we were able to achieve this with only a very small set of training data.

The most heuristic and speculative part in the design of ALICE:HWR was the choice of the features. To augment the presented HWR algorithm, this step should be less speculative. Other points of inquiry that can and should be investigated will form the next and final chapter.

7 Looking ahead

The impediment to action advances action. What stands in the way becomes the way.

— Marcus Aurelius, *Meditations V.20*

Here in this last chapter we want to list a few question that should be explored based on the considerations made in this thesis.

Question 7.1 (Parsing): *How can symbols that are built from many strokes be parsed and recognised? In particular, what is a universal mathematical model to describe the spatial relations between the strokes? And can this description unveil some of the structure of compound symbols?*

When working with symbols that are built from many stroke types, we want to know whether the recorded strokes are in the right position to actually form these symbols. We can model this, for example, by listing their positions, their sizes, their distances and how relevant the exact spatial relation between two strokes is. Fuzzy Logic has to be used in a similar fashion as for strokes. Consider, for example, the term a^x and the various ways to potentially write it seen in Figure 7.1.

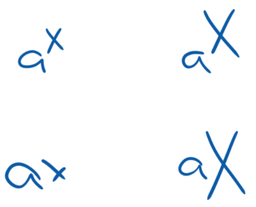


Figure 7.1: *Several ambiguous ways to write a^x or ax.*

Any recognition algorithm will have to incorporate the distance between base and exponent and their relative sizes to distinguish it from, say, `ax`.

Using all of the available data with a reasonable machine learning algorithm will certainly lead to an adequate classification process. But the question is whether this can be achieved with descriptive classification steps similar to the exclusion rules from ALICE:HWR? This is desirable, since it could lead to guided feedback the software may provide the user.

Question 7.2 (Text context): *Is there a universal and structured way to incorporate information of the context in which strokes and symbols are produced? I.e., if the application is known, how can the recognition process be improved?*

There are many examples of how recognising a single character can be affected by the context it was written in. One example we already mentioned in Section 1.3: The word `te5t` is not a word in colloquial English. So, the third character in it should probably be an `s`. Another very basic example comes from ALICE itself: the iBook uses randomised integers for the numerators and denominators in every exercise. They are usually bounded from above by 20 to focus the effort on the actual task and not complicated multiplication. So, when a pupil writes a two digit number and the HWR algorithm cannot decide whether the first digit is a `1` or a `7`, it is probably a good idea to opt for `1`.

The question is, how such considerations can be embedded into a more universal approach; without manually implementing individual rules like the last one presented. In particular, can Formal Concept Analysis provide concrete rules to make such decisions? And can Fuzzy Logic describe how likelihoods change: I.e., if we know that `1` is generally more likely than a `7` as the first digit of a number[1], how exactly have the likelihoods to adapt to account for that?

[1]by the Newcomb-Benford Law or an other analysis

Question 7.3 (More applications): *Can the ideas and methods presented here be applied directly to other applications? In particular, is it possible*

1. *to recognise sketches of geometric shapes like points, lines, circles and polytopes,*

2. *to recognise curves or gestures in 3D space?*

And, at which point does it become necessary to use additional information of the writing process like the pressure and speed of the finger/pen on the touchscreen?

Throughout this thesis, we never explicitly restricted ourselves to a certain set of symbols or characters. So, it stands to reason, that the ideas presented can be useful in any other application as long as the features are chosen well.[2] If we look at two specific applications, however, certain problems emerge.

First, the symbols in question might be much more dependent on their actual appearance; as is the case with geometric shapes. E.g., distinguishing between a `circle` and a `polygon` depends a lot on how regular the recorded stroke is. For these two forms, in particular, it is a good idea to use time and speed information: While strokes depicting a `circle` might be jaggy and irregular, they are often drawn with constant speed. In contrast, many users make a little pause at each vertex when drawing a `polygon`. So, using time steps as an additional parameter is probably a good idea.

On the other hand, as the variance in geometric shapes is usually much less as in abstract characters, they are maybe recognisable with less information. A first point to analyse is to see whether strokes depicting geometric shapes cluster more in $\mathfrak{S}$ and whether their shape vectors cluster more in $\mathbb{R}^{2(n-2)}$.

Instead of considering more planar shapes and symbols, we might consider curves in $\mathbb{R}^3$. These might represent, for example, control gestures in a virtual reality program. All basic ideas should be transferable directly. A problem here, however, is that most planar symbols are built from well-known shapes — mostly straight lines and circular arcs — in a very specific way. However, there are not many abstract icons and pictograms built from proper curves in 3D space. And the few that exist are not universally present in the mind of potential users.

[2]And [11] indicates that they are.

Question 7.4 (Negative fuzziness): *Is there a structure on the set of functions into* $[-1, 1]$ *that is compatible with, but generalises the notion of fuzzy sets?*

In Section 4.3 we saw that

$$\alpha'(x, y) = \max\{x \odot y,\ (1 - x) \odot (1 - y)\}$$

for values $x, y \in [0, 1]$ has a qualitatively similar behaviour as

$$\sigma(x, y) = \operatorname{sgn}(x) \cdot \operatorname{sgn}(y) \cdot (|x| \odot |y|)$$

for values $x, y \in [-1, 1]$. But, the latter has more symmetries and, at least for $\odot$ being the standard multiplication, it results in a differentiable function and is structurally much simpler. Concretely, $\sigma(x, y) = xy$; whereas the same function for normal fuzzy values would be

$$(x, y) \ \mapsto \ \frac{1}{2} \cdot (2x - 1)(2y - 1) + \frac{1}{2} \ = \ 2xy - x - y + \frac{1}{2},$$

if we shift fuzzy values from $[0, 1]$ to $[-1, 1]$ and the result back to $[0, 1]$. The question is: Is this an inherent property of a structure on $[-1, 1]$? A structure that mimics the way fuzzy values in $[0, 1]$ work?

Question 7.5 (Automatic learning): *Is there a way to fully automatise the training process of ALICE:HWR? In particular, is it possible for an algorithm to generate strokes from feature values and use this to perform its own attribute exploration? Moreover, what is the best way to make Algorithm 6.2.1 adapt itself to the user with minimal re-training?*

The training process of ALICE:HWR has two essential steps: Finding exclusion rules and computing idealised feature vectors for every stroke type. Ideally, the process to find both of them is fully automatic. Then, a user can train the algorithm themselves; and that with arbitrary strokes and stroke types.

To automate the search for exclusion rules, we need two components. First, a way to judge which (+)-hypotheses (and by extension which generatives) to use and how to decide whether one is more crucial during the classification process than another. Second, an idealised automatic algorithm would be able to perform attribute exploration to facilitate the search for exclusion rules. I.e., the program should, in this case, be able to produce a stroke with a given feature vector. Or, at least, with some entries of its feature vector given. Then, when the user input a few samples for a, say, 3, the algorithm could ask

Is this here also a 3?

and present the self-synthesised stroke. With the answer from the user, the algorithm could then proceed to decide which feature combinations constitute a 3 and which do not. The challenge here is that it is not obvious how to find elements in the fibres of a features.[3]

To automate the computation of idealised feature vectors, we need a way to systematically duplicate samples. In particular, we need a statistical test that is better tailored to this task and to the concrete features than the Kolmogorov-Smirnov test. Then the algorithm can reliably judge when new samples are in line with old samples.

Closely related to the question how to automatise the whole training process is how the algorithm can adapt to a user; if so desired. Both points made above have to be considered again if new samples are provided during use. In particular, as the software currently does not store the original samples, it should not store the new samples either: Ideally, it can process them directly to update both the exclusion rules and idealised feature vectors with the need to run through the initial training process in its entirety.

[3]There are several approaches to solve this problem of approximating a pseudo-inverse of $\mathfrak{S} \to [0,1]^m$ via neural networks. Here, however, we ask whether this can be done in an constructive way.

Appendix A

The Manual for the companion iBook

The interactive widgets to accompany this thesis [67] are direct applications of the statements made and allow to interact with many of the examples presented. They are designed for use on tablet computers or with graphic tablets. Using them with track pads is possible if strokes are written slowly. The use of a computer mouse is discouraged.

The basic layout for every widget is the same:

— A rectangle with a thick, black border is the area one can write in; usually only a single stroke.

— The green 'Apply'-button then triggers the desired effect. If this effect can be iterated it, it will be with about 1 iteration per millisecond. Usually, the exact speed can be adjusted.

— If an effect is applied iteratively, another click on 'Apply' stops it. If an effect is applied only once, another click on 'Apply' triggers it another time.

— The drawn strokes are rendered in a style that imitates ink. In particular, only the line segments between the recorded points are drawn. The preprocessed strokes are drawn in blue, any alterations in orange; just as in the images throughout this thesis.

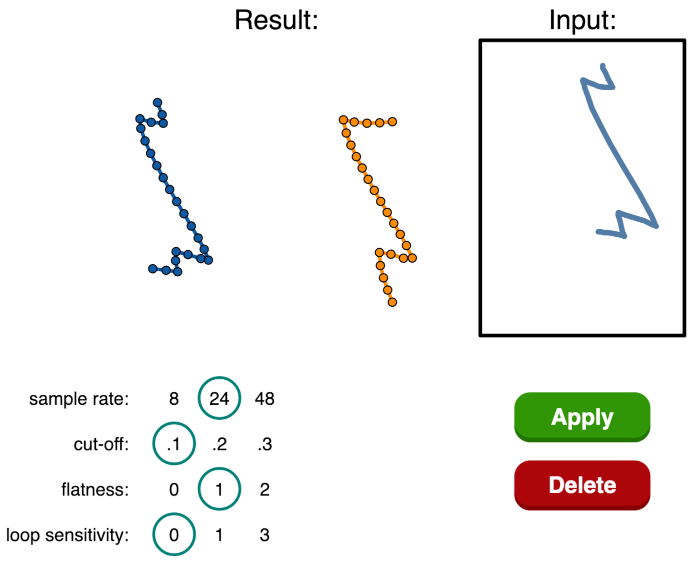
Widget A.1 (Hook removal): This widget applies the Hook Removal Algorithm 2.3.2 to a recorded stroke on both ends. Both ends of the stroke get cut off and then extrapolated again (with a flatter angle). The user can adjust the following parameters:

sample rate: The number of points n used to re-sample the recorded stroke.

cut-off: The relative number γ of points that are cut off and extrapolated again at each end of the stroke. The absolute number of points is obtained by simple rounding: $\lfloor \gamma n \rceil$.

flatness: A value f that determines how much the angle α at the last point of the stroke should get factored in when attaching the new point. The new angle created will be $\frac{\alpha}{2f}$.

loop sensitivity: The weight λ determining how much the algorithm should look for a loop at start and end. The extrapolated new stroke gets weighted with $1 - \mu_\lambda(\textsc{Start Loop}(s))$; and the original stroke s by $\mu_\lambda(\textsc{Start Loop}(s))$. In particular, a value of $\lambda = 0$ means that only the basic cut-off and extrapolation steps are performed.



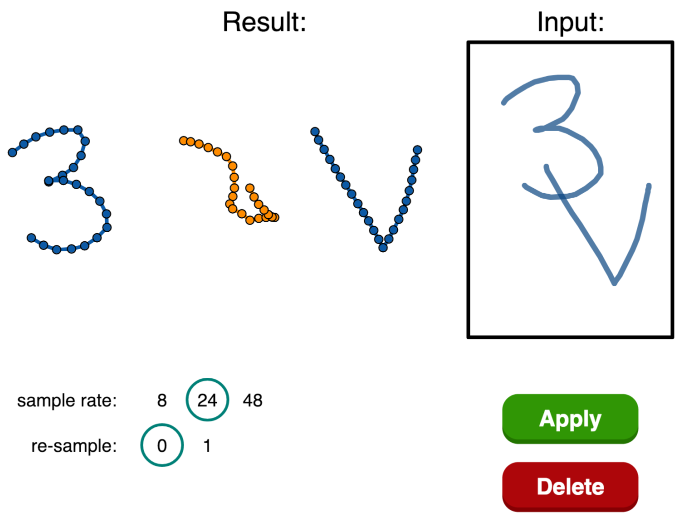
sample rate: 8 [✓ 24] 48
cut-off: [✓ .1] .2 .3
flatness: 0 [✓ 1] 2
loop sensitivity: [✓ 0] 1 3

Figure A.1: *The interface of the hook-removal widget.*

Widget A.2 (Convex combinations): This widget computes convex combinations of two recorded strokes. These strokes are depicted in blue at the left and right end of the result area. The convex combinations of them is in orange and can be manually moved from one stroke to the other. The closer the orange stroke is to one of the blue ones, the higher that blue one is weighted in the convex combination. The user can adjust the following parameters:

sample rate: The number of points n used to re-sample the recorded stroke.

re-sample: Whether or not the resulting strokes should be re-sampled. I.e., if this is set to 1, every stroke that was transformed in some way will be re-sampled.



sample rate: 8 [✓ 24] 48

re-sample: [✓ 0] 1

Figure A.2: *The interface of the convex-combination widget.*

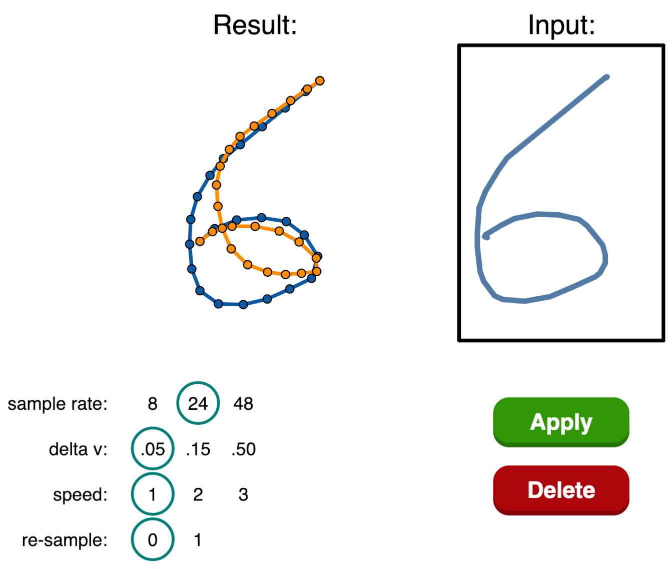
Widget A.3 (Accelerations): This widget iteratively applies an acceleration $A_{\Delta v}$ to a recorded stroke as defined in Definition 3.1.3. (This acceleration will be conjugated with a translation rotation and dilation such that the recorded strokes does not have to be normalised.) The user can adjust the following parameters:

sample rate: The number of points n used to re-sample the recorded stroke.

delta v: The parameter Δv of the accelerations. In the local coordinate system of the stroke with x-axis from start to end point and y-coordinate perpendicular to it, the x-coordinate of a point P_i gets mapped to $\frac{x_i+(i-1)\Delta v}{1+(n-1)\Delta v}$. The larger Δv, the stronger the effect. In particular, $\Delta v = 0$ would result in the identity.

speed: How often the acceleration is applied per time step. One time step is approximately one millisecond.

re-sample: Whether or not the resulting strokes should be re-sampled. I.e., if this is set to 1, every stroke that was transformed in some way will be re-sampled. This should only be used with a speed of 1 to keep the frame rate of the animation high.



sample rate: 8 [✓ 24] 48

delta v: [✓ .05] .15 .50

speed: [✓ 1] 2 3

re-sample: [✓ 0] 1

Figure A.3: *The interface of the acceleration widget.*

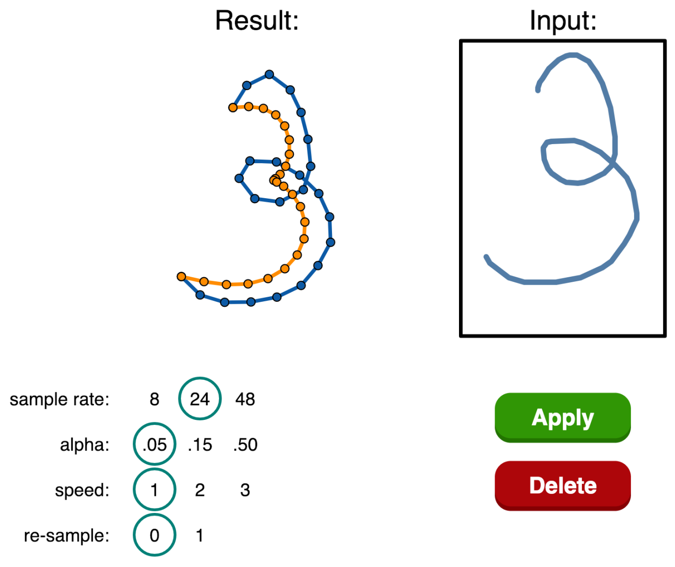
Widget A.4 (Gaussian filters): This widget iteratively applies a minimal Gaussian filter $\mathcal{G}_{A_\alpha}$ to a recorded stroke as defined in Example 3.1.9. The user can adjust the following parameters:

sample rate: The number of points n used to re-sample the recorded stroke.

alpha: The parameter α of the Gaussian filter. I.e., a point P_i on the stroke will get mapped to $\alpha P_{i-1} + (1-2\alpha)P_i + \alpha P_{i+1}$. The larger α, the stronger the effect. In particular, $\alpha = 0$ would result in the identity; and $\alpha = \frac{1}{2}$ gives the strongest possible contraction.

speed: How often the Gaussian filter is applied per time step. One time step is approximately one millisecond.

re-sample: Whether or not the resulting strokes should be re-sampled. I.e., if this is set to 1, every stroke that was transformed in some way will be re-sampled. This should only be used with a speed of 1 to keep the frame rate of the animation high.



sample rate: 8 [✓ 24] 48
alpha: [✓ .05] .15 .50
speed: [✓ 1] 2 3
re-sample: [✓ 0] 1

Figure A.4: *The interface of the Gaussian-filter widget.*

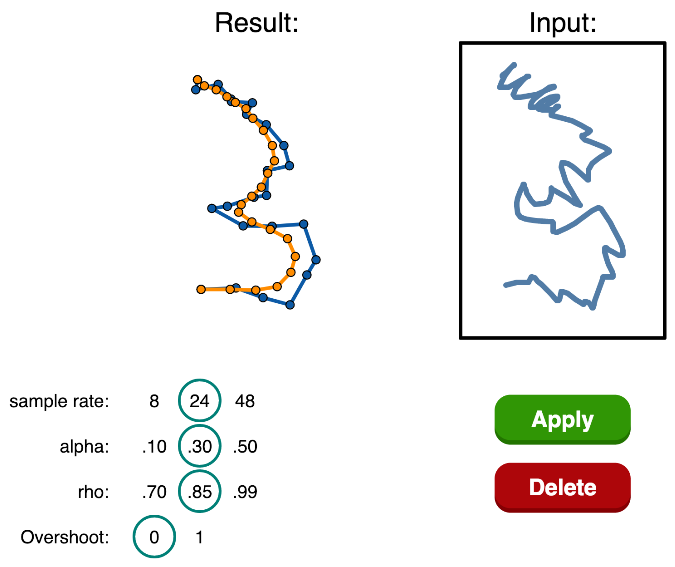
Widget A.5 (Smoothing): This widget iteratively applies the smoothing function $\mathcal{E}_{\alpha,\rho,s}$ to a recorded stroke s as defined in Example 3.2.1. It builds a convex combination of $\mathcal{G}_{A_\alpha}(s)$, a stroke smoothed by a Gaussian filter, and the original stroke s. The user can adjust the following parameters:

sample rate: The number of points n used to re-sample the recorded stroke.

alpha: The parameter α of the minimal Gaussian filter $\mathcal{G}_{A_\alpha}$ involved. I.e., a point P_i on the stroke will get mapped to $\alpha P_{i-1} + (1-2\alpha)P_i + \alpha P_{i+1}$. The larger α, the stronger the effect.

rho: The parameter ρ that weighs the image of s under the Gaussian filters compared to the original. I.e., the end result is $\rho\mathcal{G}_{A_\alpha}(s) + (1-\rho)s$. The smaller ρ, the more the intermediate result $\mathcal{G}_{A_\alpha}(s)$ gets pushed back towards s.

Overshoot: Whether $\mathcal{E}_{\alpha,\rho,s}$ is applied or $\mathcal{E}_{\alpha,\rho,2s-u}$. I.e., if this parameter is set to 1, the convex combinations that is computed at the end is $\rho\mathcal{G}_{A_\alpha}(s) + (1-\rho)(2s-u)$; where u is the straight line segment between the start and end point of s with uniformly distributed points.



sample rate: 8 [✓ 24] 48
alpha: .10 [✓ .30] .50
rho: .70 [✓ .85] .99
Overshoot: [✓ 0] 1

Figure A.5: *The interface of the smoothing widget.*

Widget A.6 (Handwriting duplications): The user can write their signature or any other similar collection of strokes into the black-bordered area. This widget then applies the geometric transformations described in Example 3.2.3 to alter this signature without changing its global appearance. The user can adjust the weights by which the three transformations are multiplied to achieve different effects.

simpler: This is the average $\frac{1}{2}\mathcal{G}^5 + \frac{1}{2}A^5$ of applying a Gaussian filter and an acceleration to reduce the details of the stroke.

cut-off: This is the average $\frac{1}{2}C_S + \frac{1}{2}C_E$ of cutting off both ends of the stroke.

projective: This is the projective transformation M that combines shearing parallel to the x-axis to the right with a "random" transformation containing a projective tilt and a rotation to the right.

(Please see the example for a precise definition of each transformations.) In the widget, the point B in the black-bordered triangle can be moved. The weights of the transformations are the barycentric coordinates of B with respect to the corners of the triangle. I.e., the closer B is to a corner the more emphasis is on that particular transformation.

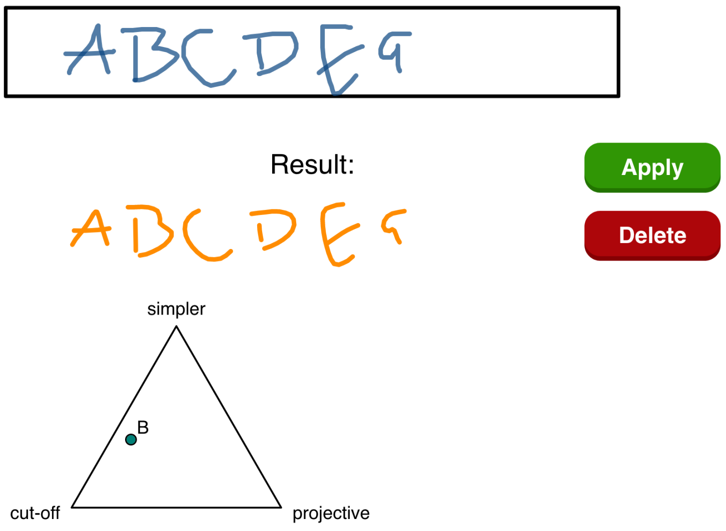


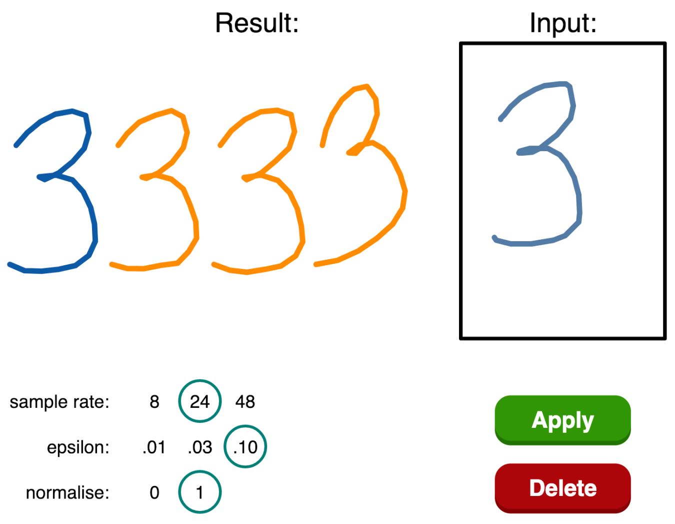
Figure A.6: *The interface of the duplication widget.*

Widget A.7 (Noise on curvature): This widget applies random noise to the points of a stroke as well as its first and second derivative; as described in Section 5.1. The results can be seen in this order from left to right. The user can adjust the following parameters:

sample rate: The number of points n used to re-sample the recorded stroke.

epsilon: The maximal distance ε a point on the stroke is allowed to move away. Note that only the distance but not the direction is controllable.

normalise: Whether the resulting strokes get normalised. I.e., when this is set to 1, the resulting strokes will be scaled and rotated such that their start and end position are in the same spatial relation than the start and end point of the recorded stroke.



sample rate: 8 [✓ 24] 48
epsilon: .01 .03 [✓ .10]
normalise: 0 [✓ 1]

Figure A.7: *The interface of the noise-on-curvature widget.*

Widget A.8 (Reconstructing strokes from determinants): This widget takes a recorded stoke s (blue) and builds another stroke t (orange) from the shape vector $\Delta(s)$ via the method given in Theorem 5.2.4. The last point of t that is seen is $\left(A_n^t\right)^{-1} b_n^s$ instead of C. As mentioned after the theorem, if the points A, B, C are at the right position indicated by the theorem, $\left(A_n^t\right)^{-1} b_n^s$ will coincide with C. The user can reposition the points A, B, C freely, and additionally adjust the following parameter:

scale: The number by which $\Delta(s)$ is multiplied before starting the construction of t. I.e., if this is set to 1, the widget illustrates Theorem 5.2.4, and if it is set to 2, the widget illustrates Corollary 5.2.5. In that case, the area of the triangle ABC has to be twice as large as $g_1(s)$ in order for $\left(A_n^t\right)^{-1} b_n^s$ and C to coincide.

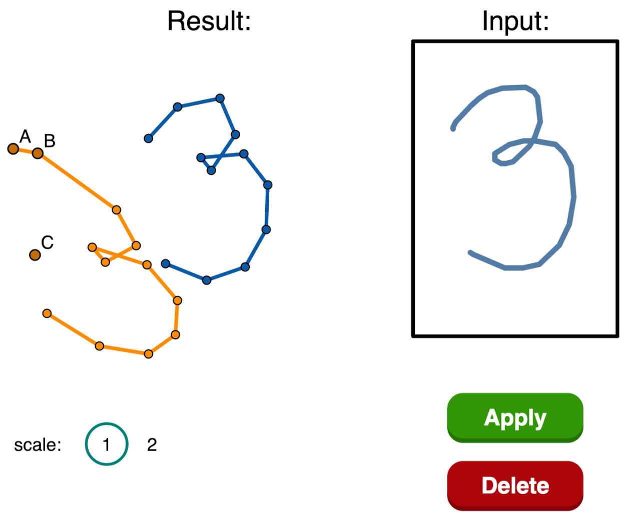


Figure A.8: *The interface of the reconstruction widget.*

Widget A.9 (ALICE:HWR): In this widget, version 4 of ALICE:HWR can be tested — the version that was used in the second study in autumn of 2017. Any numbers built from Arabic numerals written into the black-bordered rectangle will be tried to recognised. In particular, if strokes are written that obviously do not form numbers, the algorithm will still give the best guess possible. If something is unintelligible for the algorithm, the output will be three question marks.

Note that all parameters in the algorithms used by ALICE:HWR are designed around a certain size of the written strokes. That means that the recognition will usually be worse, the smaller the input strokes are. For optimal results, please aim for a height of approximately 60% to 90% of the height of the input area.

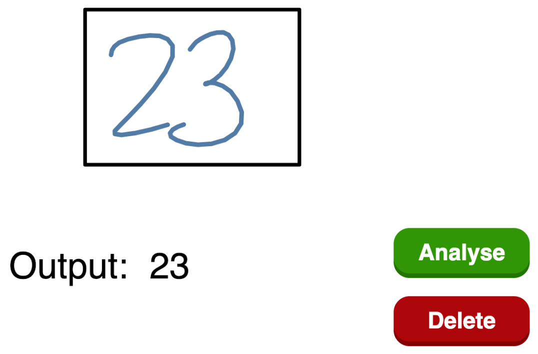


Figure A.9: *The interface of the ALICE:HWR widget.*

Appendix B

The code of ALICE:HWR

Here is a full transcript of the code used for for ALICE:HWR version 4 that was used in the 2017 ALICE study and can be tested in Widget A.9. The documentation of the individual structures and methods was left out to makes this appendix not longer than it already is. The full code can be downloaded on the same website as the companion iBook [67].

Note that there are several inconsistencies between what we presented in this thesis and the code below. This is due to the fact that the code reuses many part from previous iterations of ALICE:HWR that were not adjusted to the models and set-ups of this thesis. An example is the use of the feature Start Size which is simply $1 - $ Start Loop. So, it does not measure the likelihood that the start points is part of a loop, but the *un*likelihood of it.

All peculiarities in the code stem from practical observations and aim to improve the effectiveness of the algorithm. But none of them deviate far from the general layout presented in this thesis.

```
eps             = 0.1;
n               = 24;
preTrunc        = 2;
intense         = 3;
lambda          = 0.7;
alpha           = 0.02;
mass            = 0.65;
timeStep        = 0.5;
minGaussMatrix  = apply(1..n, i, apply(1..n, j,
    if((i == 1) % (i == n),
        if(j == i,
            1;
        , // else //
```

```
            0;
        );
    , // else //
         if(abs(j - i) == 1,
            alpha;
        ,if( j == i,
            1 - 2 * alpha;
        , // else //
            0;
        ));
    );
));
crMatrix = [[0, 1, 0, 0], [-0.5, 0, 0.5, 0], [1, -2.5, 2, -0.5], [-0.5, 1.5, -1.5,
    0.5]];

allStrokes = ["1a", "1b", "2", "3", "4a", "4b", "5a", "5b", "6", "7", "8a", "8ar", "
    8b", "9a", "9b", "0a", "0ar", "0b", "-"];
nof        = 25;

highContraRules = [
  dict(values -> [1, 1, 1, 1, 1, 1, 0.9, 0.5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1],
        label -> "1a",
         conf -> 100),

  dict(values -> [0.95, 0.2, 0.9, 0.2, 1, 1, 0.9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0.2, 1,
       1, 1, 1, 1, 1, 1, 1],
        label -> "1b",
         conf -> 100),

  dict(values -> [1, 1, 1, 1, 1, 1, 0.7, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1],
        label -> "2",
         conf -> 100),

  dict(values -> [1, 1, 1, 1, 1, 0.9, 0.9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0.8, 0.9, 1],
        label -> "3",
         conf -> 100),

  dict(values -> [1, 1, 1, 1, 1, 1, 0.9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1],
        label -> "4a",
         conf -> 100),

  dict(values -> [1, 1, 1, 1, 1, 0.8, 0.9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0.84, 1, 1,
       1, 1, 1, 1, 1],
        label -> "4b",
         conf -> 100),

  dict(values -> [0.9, 1, 0.5, 1, 1, 0.7, 0.9, 1, 1, 1, 1, 1, 1, 0.95, 1, 1, 1, 0.8,
      1, 1, 1, 1, 1, 1, 1],
        label -> "5a",
         conf -> 100),

dict(values -> [0.8, 1, 1, 0.9, 1, 0.8, 0.9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0.87,
    1, 1, 1, 1, 1, 1, 1],
        label -> "5b",
         conf -> 100),

dict(values -> [0.8, 1, 1, 1, 1, 0.8, 0 .9, 1, 0.5, 1, 1, 1, 1, 1, 1, 1, 1, 0.8,
    1, 1, 1, 1, 1, 1, 1],
        label -> "6",
         conf -> 100),

dict(values -> [1, 1, 1, 1, 1, 1, 0.9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1],
        label -> "7",
         conf -> 100),

dict(values -> [1, 1, 1, 1, 1, 0.8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1],
        label -> "8a",
         conf -> 100),

dict(values -> [1, 1, 1, 1, 1, 0.8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1],
        label -> "8ar",
         conf -> 100),

dict(values -> [1, 1, 1, 1, 1, 0.8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1],
        label -> "8b",
         conf -> 100),

dict(values -> [1, 1, 1, 1, 1, 0.8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0.8, 1, 1,
     1, 1, 1, 1],
        label -> "9a",
         conf -> 100),

dict(values -> [1, 1, 1, 1, 1, 0.8, 0.9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0.8, 1,
    1, 1, 1, 1, 1],
        label -> "9b",
         conf -> 100),

dict(values -> [1, 1, 1, 1, 1, 0.8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 0.5],
        label -> "0a",
         conf -> 100),

dict(values -> [1, 1, 1, 1, 1, 0.8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 0.5],
        label -> "0ar",
         conf -> 100),

  dict(values -> [1, 1, 1, 1, 1, 0.8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1],
        label -> "0b",
         conf -> 100),

  dict(values -> [1, 1, 1, 1, 1, 1, 0.9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1],
        label -> "-",
         conf -> 100)
];

lowContraRules = [
  dict(values -> [0, 0.6, 0, 0, 0, 0, 0, 0, 0, 0.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0],
        label -> "1a",
         conf -> 100),

  dict(values -> [0, 0, 0, 0, 0, 0.8, 0, 0, 0, 0, 0, 0, 0, 0, 0.8, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0],
        label -> "1b",
         conf -> 100),

  dict(values -> [0, 0, 0.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0],
        label -> "2",
         conf -> 100),

  dict(values -> [0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0.8, 0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0,
      0.2, 0.2, 0.2, 0.2, 0.5],
        label -> "3",
         conf -> 100),

  dict(values -> [0, 0, 0.8, 0, 0, 0, 0, 0, 0, 0, 0.8, 0, 0, 0, 0.8, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0],
        label -> "4a",
         conf -> 100),

  dict(values -> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0],
        label -> "4b",
         conf -> 100),

  dict(values -> [0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.8, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0],
        label -> "5a",
         conf -> 100),

  dict(values -> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.7, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0],
        label -> "5b",
         conf -> 100),

  dict(values -> [0, 0, 0, 0, 0, 0, 0, 0.8, 0, 0, 0, 0, 0, 0, 0.6, 0, 0, 0, 0.5, 0,
      0, 0, 0, 0, 0],
        label -> "6",
         conf -> 100),

  dict(values -> [0.8, 0.4, 0, 0, 0, 0, 0, 0, 0, 0.8, 0, 0, 0, 0, 0.6, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0],
        label -> "7",
         conf -> 100),

  dict(values -> [0, 0, 0, 0, 0, 0, 0.85, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.6,
      0, 0, 0, 0, 0.5],
        label -> "8a",
         conf -> 100),

  dict(values -> [0, 0, 0, 0, 0, 0, 0.85, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.6,
      0, 0, 0, 0, 0.5],
        label -> "8ar",
         conf -> 100),

  dict(values -> [0, 0, 0, 0, 0.3, 0, 0.85, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.6,
       0, 0, 0, 0, 0],
        label -> "8b",
         conf -> 100),

  dict(values -> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.7, 0, 0, 0,
      0, 0, 0, 0],
        label -> "9a",
         conf -> 100),

  dict(values -> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.7, 0, 0, 0,
      0, 0, 0, 0],
        label -> "9b",
         conf -> 100),

  dict(values -> [0, 0, 0, 0, 0, 0, 0.85, 0.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0],
        label -> "0a",
         conf -> 100),

  dict(values -> [0, 0, 0, 0, 0, 0, 0.85, 0, 0.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0],
        label -> "0ar",
         conf -> 100),

  dict(values -> [0, 0, 0.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.8, 0, 0, 0,
       0, 0, 0, 0],
        label -> "0b",
         conf -> 100),

  dict(values -> [0.5, 0, 0.8, 0, 0, 0.9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0],
        label -> "-",
         conf -> 100)
];

fuzzyContext = [
  dict(label -> "1a",
       feats -> [1, 0.8, -0.2, -1, 0.8, 0.2, -0.2, -1, 1, 1, -1, 1, -1, -1, -0.2,
           0.8, -1, -0.8, -0.8, 0, 0.2, 0.8, 0.8, 0.4, 0]),
  dict(label -> "1b",
       feats -> [0, -1, 0, -1, -1, 1, -1, 0, 0, 0, 0, 0.5, 0.5, 0, 1, 0, -1, -1, -1,
            1, -0.8, -0.5, -0.5, -0.2, 0]),
  dict(label -> "2",
       feats -> [1, 0.4, 1, -0.4, -1, 0.2, -0.2, -0.6, 0.6, 0, 0, 1, -1, -1, 0.8, 1,
            -1, -0.4, 0.6, 0.8, 0.6, 0.2, -0.2, 0.4, -1]),
  dict(label -> "3",
       feats -> [1, -0.2, -1, -0.6, 1, 0, 0, -0.8, 0.8, 1, -1, 0.8, 0.4, -1, 1, -1,
           -1, -0.2, -0.2, 0.8, 0.6, 0.8, 0.2, 0, 1]),
  dict(label -> "4a",
       feats -> [-0.6, -1, 1, 0.2, 0.8, 0, 0, 1, -1, -1, 1, 1, -0.6, -0.6, 1, 1, -1,
            -0.8, -0.8, 0.2, 0.2, -0.8, -0.8, 0.4, -1]),
  dict(label -> "4b",
       feats -> [-1, -0.4, -0.2, -1, -0.4, -0.6, 0.6, -1, 1, -1, 1, -1, 1, 1, -0.8,
           -0.8, -1, 0.4, 0.2, 0.8, 0.2, -0.8, 0.8, 0.2, -1]),
  dict(label -> "5a",
       feats -> [0, -0.8, -1, -0.8, -0.8, 0.3, -0.3, -0.6, 0.6, 0.6, -0.6, 0.5, 0.5,
            -0.5, 1, -1, -1, 0.6, -0.5, 1, 0.6, 0.8, 0, 0.2, -1]),
  dict(label -> "5b",
       feats -> [-1, 0, -1, -0.4, -1, 0.2, -0.2, -0.2, 0.2, -0.2, 0.2, -0.8, 1, 1,
           1, -1, -1, -1, 0.2, 1, 0.4, 0, 0.2, 0.2, -1]),
  dict(label -> "6",
       feats -> [-1, -0.6, -1, 0.2, -0.8, -0.4, 0.4, 1, -1, 0, 0, -0.8, 1, 1, 1, -1,
            -1, -0.4, 1, 0.8, 0.2, -0.8, -0.8, 0, -1]),
  dict(label -> "7",
       feats -> [1, 0.2, -0.6, -1, 0.8, 0.2, -0.2, -0.2, 0.2, 1, -1, 1, -0.6, -1, 1,
            -0.8, -1, -0.8, -0.8, 0.5, 0.2, 0.8, 0.6, 0.4, -1]),
  dict(label -> "8a",
       feats -> [-1, -0.6, -1, 0.6, -0.8, -1, 1, -0.1, 0.1, 0, 0, 0.4, -1, -0.8, 1,
           -0.4, 1, 1, 1, 1, 0.8, -0.2, -1, 0.4, 1]),
  dict(label -> "8ar",
       feats -> [1, -0.6, 1, 0.6, -0.8, -1, 1, 0.1, -0.1, 0, 0, -1, 0.4, -0.8, 1,
           -0.4, 1, 1, 1, 1, 0.8, -0.2, -1, 0.4, 1]),
  dict(label -> "8b",
       feats -> [-0.5, -0.1, 0.5, 0.7, 0.9, -1, 1, -0.1, 0.1, -1, 1, 0.4, -1, -0.8,
           1, -0.4, 1, 1, 1, 1, 0.8, -0.2, -1, 0.4, -1]),
  dict(label -> "9a",
       feats -> [-1, -0.6, -1, -0.8, -1, 0.6, -0.6, -0.2, 0.2, 0.4, -0.4, -0.2, 1,
          0.4, 1, -1, -1, 1, -0.2, 0.8, 0.6, 0.6, 0.2, 0, -1]),
  dict(label -> "9b",
       feats -> [-1, -0.6, -0.8, -1, -1, -0.4, 0.4, -1, 1, -0.6, 0.6, -0.6, 1, 0.4,
           0, -1, -1, 1, -0.4, 0.8, 0.6, 0.6, 0.8, 0, -1]),
  dict(label -> "0a",
       feats -> [-1, -0.5, -1, 0.6, -0.8, -1, 1, 1, -1, 0, 0, -0.4, 0, -0.6, 1,
           -0.8, 0.8, 1, 1, -0.2, 0.8, 0, -1, 0.8, -1]),
  dict(label -> "0ar",
       feats -> [1, -0.5, 1, 0.6, -0.8, -1, 1, -1, 1, 0, 0, -0.4, 0, -0.6, 1, -0.8,
           0.8, 1, 1, -0.2, 0.8, 0, -1, 0.8, -1]),
  dict(label -> "0b",
       feats -> [-1, -0.8, 1, 0.2, 1, -0.8, 0.8, 0.8, -0.8, -1, 1, 0.6, -1, -0.8, 1,
            1, 1, 1, 1, -0.6, 0.8, -0.4, -1, 0.8, -1]),
  dict(label -> "-",
       feats -> [1, 0, 1, 0.2, -1, 1, -1, 0, 0, 0, 0, 0.6, -1, -1, -0.4, 1, 0.4, -1,
            -1, 1, -0.8, 0.2, 0, -0.2, -1])
];

scoreContext = apply(1..length(allStrokes),
  dict(label -> allStrokes_#, feats -> 0.5 * round(get(fuzzyContext_#, "feats")) +
      (1 - 0.5) * get(fuzzyContext_#, "feats"));
);

crystalPool = [
  dict(label -> "1", strokes -> ["1a"]),
  dict(label -> "1", strokes -> ["1b"]),
  dict(label -> "2", strokes -> ["2"]),
  dict(label -> "3", strokes -> ["3"]),
  dict(label -> "4", strokes -> ["4a", "1b"]),
  dict(label -> "4", strokes -> ["4b"]),
  dict(label -> "5", strokes -> ["5a", "-"]),
  dict(label -> "5", strokes -> ["5b"]),
  dict(label -> "5", strokes -> ["5b", "-"]),
  dict(label -> "5", strokes -> ["9a", "-"]),
  dict(label -> "5", strokes -> ["3", "-"]),
  dict(label -> "6", strokes -> ["6"]),
  dict(label -> "7", strokes -> ["7"]),
  dict(label -> "7", strokes -> ["1a", "-"]),
  dict(label -> "7", strokes -> ["7", "-"]),
  dict(label -> "8", strokes -> ["8a"]),
  dict(label -> "8", strokes -> ["8ar"]),
  dict(label -> "8", strokes -> ["8b"]),
  dict(label -> "9", strokes -> ["9a"]),
  dict(label -> "9", strokes -> ["9b"]),
  dict(label -> "0", strokes -> ["0a"]),
  dict(label -> "0", strokes -> ["0ar"]),
  dict(label -> "0", strokes -> ["0b"])
];

between(a, c, d) := (a >= c) & (a <= d);

```

```
binom(m, k) := (
    if((m < 0) % (k < 0) % (k > m),
        err("binom: wrong numbers")
    , // else //
        faculty(m) / faculty(k) / faculty(m - k)
    );
);

bite(list, i) := list_((i + 1)..length(list));
bite(list) := bite(list, 1);

box(list) := (
    regional(bl, diag);

    bl   = min(list);
    diag = max(list) - bl;
    expandrect(bl, diag.x, diag.y);
);

computeangle(p, q, r) := (
    regional(x, y, s, w);

     x = p - q;
     y = r - q;
     s = (x * y) / (abs(x) * abs(y));
     s = if(s < -1, -1, if(s > 1, 1, s));
     w = arccos(s) + 0;

     if(perp(x) * y >= 0, w, 2*pi - w);
);

consectriples(list) := (
    regional(res);

    res = [];
    if(length(list) <= 2,
        res = [];
    , // else //
        forall(1..(length(list) - 2),
            res = res :> list_[#, # + 1, # + 2];
        );
    );
);

const(m, x) := if(m == 0, [], apply(1..m, x));

expandrect(pos, c, w, h) := (
    regional(d, e, shift);

    d     = 0.5 * [w, h];
    e     = (d_1, -d_2);
    shift = compass()_c;
    shift = (0.5 * w * shift.x, 0.5 * h * shift.y);
    apply([-d, e, d, -e], pos + # + shift);
);
expandrect(pos, w, h) := expandrect(pos, 1, w, h);
compass() := apply(directproduct([1, 0, -1], [1, 0, -1]), reverse(#));

faculty(m) := if(m <= 0, 1, m * faculty(m - 1));

findin(list, x) := (
    regional(occs);

    occs = select(1..length(list), list_# == x);
    if(length(occs) == 0, 0, occs_1);
);

frequency(list, x) := length(select(list, # == x));

intersect(a, b) := (
      area(a_1, a_2, b_1) * area(a_1, a_2, b_2) < 0
    & area(b_1, b_2, a_1) * area(b_1, b_2, a_2) < 0
);

isconst(list) := (
         list == const(length(list), list_1);
);
isconst(list, x) := (
         list == const(length(list), x);
);

findperm(list1, list2) := (
    apply(list2, e, select(1..length(list1), list1_# == e)_1);
);

pop(list) := list_(1..(length(list) - 1));
pop(list, i) := list_(1..(length(list) - i));

randchoose(list) := list_(randomint(length(list)) + 1);

randomindex(n) := (
    regional(start, res, r);

        start = 1..n;
        res   = [];
        forall(1..n,
                r     = randomint(n - # + 1) + 1;
                res   = res :> start_r;
                start = start -- res;
        );
        res;
);
```

```

randsort(list) := list_(randomindex(length(list)));

triangleheight(a, b, x) := if(a ~= b, dist(x, a), 2 * det(a :> 1, b :> 1, x :> 1) /
    dist(a, b));

zip(x, y) := transpose([x, y]);


catmullRomPiece(p0, p1, p2, p3, t) := (
  [1, t, t^2, t^3] * crMatrix * [p0, p1, p2, p3]
);

circ(a, b, c) := (
    regional(m1, m2, p1, p2, mid, dist);

    m1   = (a + b) / 2;
    m2   = (b + c) / 2;
    p1   = perp(join(a, b), m1);
    p2   = perp(join(b, c), m2);
    mid  = meet(p1, p2);
    dist = if(mid.homog_3 == 0, 10000000, |mid.xy, a.xy|);
    (mid, dist);
);

continuestrokeOLD(list, mode, steps) := (
    regional(indices, current);

    indices = if(mode == "start",
        1
    , // else //
        if(mode == "end",
            -1
        , // else //
            err("continuestroke: wrong mode");
            0
        );
    ) * reverse(1..steps);
    current = list_indices;
    sum(apply(1..steps, k, (-1)^(steps - k) * binom(steps, k - 1) * current_k));
);

continuestroke(list, mode) := (
  regional(v, w, alpha, res);

  if(mode == "end",
    reverse(continuestroke(reverse(list), "start"));
  , // else //
    v     = list_(-2) - list_(-3);
    w     = list_(-1) - list_(-2);
    alpha = 0.5 * if(det(list_(-3), list_(-2), list_(-1)) > 0, 1, -1) * arccos(v * w
         / abs(v) / abs(w));

    res   = list_(-1) + [[cos(alpha), -sin(alpha)], [sin(alpha), cos(alpha)]] * w *
        abs(w) / abs(v);

    list :> res;
  );
);

continuestroke(list, mode, its) := (
  if(its == 1,
    continuestroke(list, mode);
  , // else //
    continuestroke(continuestroke(list, mode), mode, its - 1);
  );
);

matPats(list, eps) := (
    regional(flag, counter);

    flag    = false;
    counter = 0;
    forall(2..(length(list) - 1),
        if((list_(# - 1)).x < (list_#).x + eps & (list_(# + 1)).x < (list_#).x + eps
            ,
            if(!flag, counter = counter + 1);
            flag = true;
        , // else //
            flag = false;
        );
    );

    counter;
);
matPats(list) := matPats(list, 0.001);

derive(list) := apply(consecutive(list), #_2 - #_1);

diameter(list) := max(apply(pairs(list), dist(#_1, #_2)));

dimensions(list) := max(list) - min(list);

direction(list) := (list_(-1) - list_1) / dist(list_1, list_(-1));

equidist(list, nop) := (
  regional(dists, traj, before, after, cutTimes, piece, res);

  dists    = apply(derive(list), abs(#));
  traj     = sum(dists);
  before   = 2 * list_1    - list_2;
  after    = 2 * list_(-1) - list_(-2);
  cutTimes = 0 <: apply(1..(length(dists) - 1), sum(dists_(1..#))) / traj;

  res = apply(0..(nop - 1), i,
    piece = select(1..(length(list) - 1), cutTimes_# * (nop - 1) <= i)_(-1);

    if(piece == 1,
      catmullRomPiece(before, list_1, list_2, list_3, i / (nop - 1) * traj / dists_1
          );
    ,if(piece == length(list) - 1,
      catmullRomPiece(list_(-3), list_(-2), list_(-1), after, (i / (nop - 1) -
          cutTimes_(-1)) * traj / dists_(-1));
    , // else //
      catmullRomPiece(list_(piece - 1), list_(piece), list_(piece + 1), list_(piece
          + 2), (i / (nop - 1) - cutTimes_piece) * traj / dists_piece);
    ));
  );
);

resizesymbol(list) := (
    regional(bbox, scale, center);

    bbox   = box(flatten(list));
    if(dist(bbox_1, bbox_4) > 0.2,
        scale  = 10 / dist(bbox_1, bbox_4);
        center = 0.5 * (bbox_1 + bbox_3);
        list   = apply(list, s, apply(s, # - center));
        apply(list, s, apply(s, center + scale * #));
    , // else //
        list;
    );
);

trajectory(list) := sum(apply(derive(list), abs(#)));

gauss(list, matrix) := (
    apply(1..length(list), i,
        sum(apply(1..length(list), j,
            matrix_i_j * list_j;
        ));
    );
);
gauss(list, matrix, iterations) := (
    regional(res);

    res = list;
    repeat(iterations,
        res = gauss(res, matrix);
    );
);

startDir(list) := (
  regional(dir);

  dir = list_3 - list_1;
  dir / abs(dir);
);
endDir(list) := (
  regional(dir);

  dir = list_(-1) - list_(-3);
  dir / abs(dir);
);
genDir(list) := (
  regional(dir);

  dir = list_(-1) - list_1;
  dir / abs(dir);
);

lineness(list) := dist(list_1, list_(-1)) / trajectory(list);

offOfSE(list) := (
  regional(res);
  res = apply(list, triangleheight(list_1, list_(-1), #));

  res = select(res, abs(#) > 0.1);
  if(length(res) == 0,
    [0.5, 0.5];
  , // else //
    [length(select(res, # > 0)), length(select(res, # < 0))] / length(res);
  );
);

curvature(list) := (
    regional(r, dir1, dir2, res, steps);

    steps = 2;

  res = apply((1 + steps)..(length(list) - steps),
    triangleheight(list_(# - steps), list_(# + steps), list_#);
  );

  res = select(res, abs(#) > 0.03);
  if(length(res) == 0,
      [0.5, 0.5];
  , // else //
      [length(select(res, # > 0.03)), length(select(res, # < - 0.03))] / length(res)
          ;
  );
);

awayfromstart(list) := (
    regional(sortedlist, ended, counter);

    sortedlist = sort(list, dist(list_1, #));

    ended   = false;
    counter = 0;
    forall(1..length(list),
        if(!ended & (list_# == sortedlist_#),
            counter = counter + 1;
        , // else //
            ended = true;
        );
    );
    counter;
);

startsize(list) := (
  regional(close);

  close = sort(max(awayfromstart(list), 3)..length(list), dist(list_1, list_#));
  close = close_1;

  dist(list_1, list_close) / diameter(list);
);
endsize(list) := startsize(reverse(list));

relativestart(list) := (
    regional(dims, pointer);

    dims    = 0.5 * dimensions(list);
    dims    = apply(dims, if(# ~= 0, 10, #));
    pointer = list_1 - 0.5 * (max(list) + min(list));
    [pointer_1 / dims_1, pointer_2 / dims_2];
);
relativeend(list) := (
    regional(dims, pointer);

    dims    = 0.5 * dimensions(list);
    dims    = apply(dims, if(# ~= 0, 1000, #));
    pointer = list_(-1) - 0.5 * (max(list) + min(list));
    [pointer_1 / dims_1, pointer_2 / dims_2];
);

scaleTo01(x) := 0.5 * x + 0.5;
cutOffNeg(x) := if(x < 0, 0, x);

rescale(c, l, x) := (
  if(x <= l,
    l * x / ((1 - c) * x + c * l);
  , // else //
    ((x - l) * (c - l) + l - l^2) / ((c - 1) * (x - l) + 1 - l);
  );
);


modProb(x, c) := c * x / ((c - 1) * x + 1);

truncateStart(list, relSize) := equidist(bite(list, floor(relSize * n)), n);
truncateEnd(list, relSize)   := equidist(pop(list, floor(relSize * n)), n);

preprocessing(list) := (
  regional(gaussed, new, s, e, core);

  list = bite(list);

  if(length(list) < 4,
    list = [list_1, 0.75 * list_1 + 0.25 * list_2, 0.25 * list_(-2) + 0.75 * list_
        (-1), list_(-1)];
  );

  list = apply(list, # + 0.01 * eps * [random(), random()]);
  list = equidist(list, n);

  // Noise reduction
  gaussed  = gauss(list, minGaussMatrix, 2);

  // Hook removal
  core = gaussed_((1 + preTrunc)..(n - preTrunc));
  new  = continuestroke(continuestroke(core, "start", preTrunc), "end", preTrunc);

  // Merging
  s = modProb(startsize(gaussed), 5);
  e = modProb(endsize(gaussed), 5);

  res = s * new_(1..preTrunc) + (1 - s) * gaussed_(1..preTrunc) ++ core ++ e * new_
      ((-preTrunc)..(-1)) + (1 - e) * gaussed_((-preTrunc)..(-1));
);

cross(o, a, b) := (
  (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x)
);
convexHull(points) := (
  regional(ordered, upper, lower);

  ordered = set(sort(points));
  if(length(ordered) <= 3,
    ordered;
  , // else //
    lower = [];
    forall(ordered,
      while((length(lower) > 1) & (cross(lower_(-2), lower_(-1), #) <= 0),
        lower = pop(lower);
      );
      lower = lower :> #;
    );
    upper = [];
    forall(reverse(ordered),
      while((length(upper) > 1) & (cross(upper_(-2), upper_(-1), #) <= 0),
        upper = pop(upper);
      );
      upper = upper :> #;
    );

    pop(lower) ++ pop(upper);
  );
);
areaOfPolygon(points) := (
  0.5 * abs(sum(apply(1..(length(points) - 1),
    (points_#).x * (points_(# + 1)).y - (points_(# + 1)).x * (points_#).y;
  )));
);

analyzeStroke(list) := (
  regional(dir1, dir2, dir3, dir4, dir5, curv, hull, middle, traj, cog, distsA,
      distsB, center);

  dim     = dimensions(list);
  center  = 0.25 * sum(box(list));
  dir1    = startDir(list);
  dir2    = endDir(list);
  dir3    = genDir(list);
  dir4    = relativestart(list);
  dir5    = relativeend(list);
  curv    = curvature(list);
  hull    = convexHull(list);
  traj    = trajectory(list);
  middle  = list_(select(2..length(list), 2 * trajectory(list_(1..#)) >= traj)_1);
  cog     = sum(list) / n;
  distsA  = apply(list, dist(#, cog));
  distsB  = apply(list, dist(#, center));

  apply([
      scaleTo01(dir1.x),
      scaleTo01(dir1.y),
      scaleTo01(dir2.x),
      scaleTo01(dir2.y),

      scaleTo01(-dir1 * dir2),

      lineness(list),
      1 - lineness(list),

      if(curv_2 < 0.2, 0, curv_2),
      if(curv_1 < 0.2, 0, curv_1),

      offOfSE(list)_1,
      offOfSE(list)_2,

      cutOffNeg(dir3 * (1,-1) / sqrt(2)),
      cutOffNeg(dir3 * (-1,-1) / sqrt(2)),

      scaleTo01(dir4.x),
      scaleTo01(dir4.y),
      scaleTo01(dir5.x),
      scaleTo01(dir5.y),

      1 - startsize(list),
      1 - endsize(list),

      1 - min(distsB) / abs(0.5 * dim),

      areaOfPolygon(hull :> hull_1) / (dim.x * dim.y),

      scaleTo01((middle.x - 0.5 * ((list_1).x + (list_(-1)).x)) / dim.x),
      scaleTo01((middle.y - 0.5 * ((list_1).y + (list_(-1)).y)) / dim.y),

      sum(distsA) / n / max(distsA),

      if(matPats(list) >= 2, 1, 0)
    ],
      rescale(intense, lambda, #);
    );
);

classify(recFeatlist) := (
  regional(proMet, contraMetH, contraMetL, contraResH, contraResL, allLeftovers,
      scores);

  proMet = [];

  if(length(proMet) == 1,
    [get(proMet_1, "label"), 1000];
  , // else //
    if(length(proMet) > 1,
      scores = apply(recFeatlist, totalScore(apply(proMet, get(#, "label")), #));
      sort(collectScores(scores), #_2)_(-1);
    , // else //
      contraMetH = highContraRulesMet(recFeatlist_1);
      contraMetL = lowContraRulesMet(recFeatlist_1);

      contraMetH = apply(contraMetH, get(#, "label"));
      contraMetL = apply(contraMetL, get(#, "label"));

      contraResH = allStrokes;
      forall(reverse(contraMetH),
        if((contraResH -- [#]) != [],
          contraResH = contraResH -- [#];
        );
      );
      contraResL = allStrokes;
      forall(reverse(contraMetL),
        if((contraResL -- [#]) != [],
          contraResL = contraResL -- [#];
        );
      );
      allLeftovers = (contraResH ~~ contraResL);

      if(length(allLeftovers) >= 1,
        if(length(allLeftovers) == 1,
          [(allLeftovers)_1, 1000];
        , // else //
          scores = apply(recFeatlist, totalScore(allLeftovers, #));
          sort(collectScores(scores), #_2)_(-1);
        );
      , // else //
        if(length(contraResH ++ contraResL) >= 1,
          scores = apply(recFeatlist, totalScore(contraResH ++ contraResL, #));
          sort(collectScores(scores), #_2)_(-1);
        , // else //
          scores = apply(recFeatlist, totalScore(allStrokes, #));
          sort(collectScores(scores), #_2)_(-1);
        );
      );
    );
  );
);
highContraRulesMet(recFeat) := (
  select(highContraRules, r,
    length(select(1..nof, recFeat_# > get(r, "values")_#)) > 0;
  );
);
lowContraRulesMet(recFeat) := (
  select(lowContraRules, r,
    length(select(1..nof, recFeat_# < get(r, "values")_#)) > 0;
  );
);
collectScores(list) := (
  regional(labels);

  list   = flatten(list);
  labels = set(apply(list, #_1));

  apply(labels, l, [l, sum(apply(select(list, #_1 == l), p, p_2))]);
);
totalScore(pool, score) := (
    sort(angleScore(pool, score), #_1);
);
angleScore(pool, recFeat) := (
  regional(scores, res, singles, base);
  pool   = select(scoreContext, contains(pool, get(#, "label")));
  scores = apply(pool,
    base = apply(get(#, "feats"), f, scaleTo01(f));
    [get(#, "label"), abs(base * recFeat / abs(base) / abs(recFeat))]
  );

  scores;
);

orienting(list) := (
  regional(forwards, backwards, iterates);

  forwards  = analyzeStroke(list);
  backwards = analyzeStroke(reverse(list));

  if(sort(totalScore(allStrokes, forwards), #_2)_(-1)_2 >= sort(totalScore(
      allStrokes, backwards), #_2)_(-1)_2 - 0.1,
    [0.01 * round(100 * forwards)];
  , // else //
    [0.01 * round(100 * backwards)];
  );
);

parse(atoms, crysPool) := (
  regional(halves, potPart, pairCrystals, defPart, pairs, singles, c);

  atoms  = sort(atoms, (#_1).x);
  atoms  = apply(1..length(atoms), # <: atoms_#);
  halves = select(atoms, contains(["4a", "5a", "-"], #_3));

  pairCrystals = apply(crysPool, sort(get(#, "strokes")));
  pairCrystals = select(pairCrystals, length(#) == 2);

  potPart = apply(halves, h,
    select([h_1 - 1, h_1 + 1], i,
      contains(pairCrystals, sort([h_3, atoms_i_3]))
    );
  );

  if(contains(potPart, []),
    [[0, "???"]];
  , // else //
    defPart = apply(1..length(halves), i,
      sort(potPart_i, dist((halves_i_2).x, (atoms_#_2).x))_1;
    );

    if(length(set(defPart)) != length(halves),
      [[0, "???"]];
    , // else //
      pairs = set(apply(1..length(halves), i,
        c = select(crysPool, sort(get(#, "strokes")) == sort([halves_i_3, atoms_(
            defPart_i)_3]))_1;
        [sort([halves_i_1, defPart_i]), get(c, "label")];
      ));

      singles = (1..length(atoms)) -- flatten(apply(pairs, #_1));
      singles = apply(singles, s,
        c = select(crysPool, get(#, "strokes") == [atoms_s_3])_1;
        [[s], get(c, "label")]);

      sort(singles ++ pairs, #_1_1);
    );
  );
);

processdata(list) := (
  regional(classStrokes, foundCrys, res);

  list = apply(list, if(length(#) <= 1, 42, #)) -- [42];
  list = apply(list, preprocessing(#));

  classStrokes = apply(list,
    res = classify(orienting(#));
    [sum(#) / n, res_1, res_2];
  );

  foundCrys = parse(classStrokes, crystalPool);

  sum(apply(foundCrys, text(#_2)));


);

addToRecord(list) := (
  if(dist(mouse().xy, list_(-1)_(-1)) > eps,
      if(length(list_(-1)) < 2,
        list_(-1) = list_(-1) :> mouse().xy;
      , // else //
        list_(-1) = list_(-1) :> (0.5 * (mouse().xy - list_(-1)_(-1)) / mass *
            timeStep^2
                                  + (list_(-1)_(-1) - list_(-1)_(-2)) * timeStep
                                  + list_(-1)_(-1));
      );
  );
  list;
);
```

Bibliography

[1] Apple. *iOS Device Compatibility Reference*. URL: `https://developer.apple.com/library/archive/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/Displays/Displays.html`.

[2] Nur Sukinah Aziz, Mohd Nizam Saad, Abd. Hadi Abd. Razak and Azman Yasin. 'Redesigning the User Interface of Handwriting Recognition System for Preschool Children'. In: *Proceedings of the 2nd International Conference on Education Technology and Computer*. 2010, pp. 393–404.

[3] Drew H. Bailey, Mary K. Hoard, Lara Nugent and David C. Geary. 'Competence with fractions predicts gains in mathematics achievement'. In: *Journal of Experimental Child Psychology* 113.3 (2012), pp. 447–455.

[4] John Perry Barlow. 'A Declaration of the Independence of Cyberspace'. In: *Electronic Frontier Foundation* (1996).

[5] Roy F. Baumeister, Ellen Bratslavsky, Mark Muraven and Dianne M. Tice. 'Ego depletion: Is the active self a limited resource?' In: *Journal of Personality and Social Psychology* 74 (1998), pp. 1252–1265.

[6] Edwin Catmull and Raphael Rom. 'A Class of Local Interpolating Splines'. In: *Computer Aided Geometric Design*. Academic Press, 1974.

[7] Zheru Chi, Hong Yan and Tuan Pham. *Fuzzy Algorithms: With Applications to Image Processing and Pattern Recognition*. World Scientific, 1996.

[8] Dan C. Cireşan, Ueli Meier and Jürgen Schmidhuber. 'Multi-column Deep Neural Networks for Image Classification'. In: *CoRR* abs/1202.2745 (2012).

[9] Andy Clark. 'An embodied cognitive science?' In: *Trends in Cognitive Sciences* 3.9 (1999), pp. 345–351.

[10] Mihály Csíkszentmihályi. *Flow: The Psychology of Optimal Experience*. Harper & Row, 1990.

[11] Adrien Delaye and Eric Anquetil. 'HBF49 feature set: A first unified baseline for online symbol recognition'. In: *Pattern Recognition* 46 (2013), pp. 117–130.

[12] Max Frenkel and Ronen Basri. 'Curve Matching Using the Fast Marching Method'. In: *EMMCVPR 2003, LNCS 2683*. Springer, 2003, pp. 35–51.

[13] Martin von Gagern, Ulrich Kortenkamp, Jürgen Richter-Gebert and Michael Strobel. 'CindyJS'. In: *Mathematical Software – ICMS 2016, Proceedings of the 5th International Conference*. https://github.com/cindyjs. Berlin, Germany: Springer, 2016, pp. 319–326.

[14] Bernhard Ganter and Sergei Kuznetsov. 'Formalizing Hypotheses with Concepts'. In: *Proceedings of 8th International Conference on Conceptual Structures*. Darmstadt, Germany: Springer, 2000, pp. 342–356.

[15] Bernhard Ganter and Rudolf Wille. *Formale Begriffsanalyse*. German. Springer, 1996.

[16] Qin Gao, Bin Zhu, Pei-Luen Patrick Rau, Shilpa Vyas, Cuiling Chen and Hui Li. 'User Experience with Chinese Handwriting Input on Touch-Screen Mobile Phones'. In: *Cross-Cultural Design. Methods, Practice, and Case Studies*. Ed. by P. L. Patrick Rau. Springer, 2013, pp. 384–392.

[17] R. L. Graham. 'An efficient algorithm for determining the convex hull of a finite planar set'. In: *Information Processing Letters* 1 (1972), pp. 132–133.

[18] Beki Grinter. *What is Interactive Computing?* URL: `https://beki70.wordpress.com/2011/01/27/what-is-interactive-computing/`.

[19] J.-L. Guigues and Vincent Duquenne. 'Familles minimales d'implications informatives resultant d'un tableau de donnees binaires'. French. In: *Math. Sei. Humaines* 95 (1986), pp. 5–18.

[20] John Hattie, ed. *Visible Learning. A Synthesis of over 800 Meta-Analyses relating to Achievement*. Routledge, 2009.

[21] Delia Hillmayr, Lisa Ziernwald, Frank Reinhold, Sarah I. Hofer and Kristina Reiss. 'The Effectiveness of Learning Mathematics and Sciences with Digital Media in Secondary Schools: A Comprehensive Meta-Analysis'. In: (2018). manuscript submitted for publication.

[22] Stefan Hoch. 'Prozessdaten aus digitalen Schulbüchern (working title)'. German. PhD thesis. Technische Universität München, planned for 2019.

[23] Stefan Hoch, Frank Reinhold, Bernhard Werner, Kristina Reiss and Jürgen Richter-Gebert. 'Interactive Textbooks: The Case of Fractions'. In: *II International Conference on Mathematics Textbooks Research and Development (ICMT2 2017)*. Rio de Janeiro, 2017.

[24] Stefan Hoch, Frank Reinhold, Bernhard Werner, Kristina Reiss and Jürgen Richter-Gebert. 'Prozessdatenanalysen: Darstellung von Brüchen'. In: *Beiträge zum Mathematikunterricht 2017*. Ed. by Ulrich Kortenkamp and A. Kuzle. Münster: WTM-Verlag, 2017, pp. 424–428. DOI: `10.17877/DE290R-18527`.

[25] Stefan Hoch, Frank Reinhold, Bernhard Werner, Kristina Reiss and Jürgen Richter-Gebert. *Bruchrechnen. Bruchzahlen & Bruchteile greifen & begreifen*. Deutsche Apple iBooks Version. München: Technische Universität München, 2018. URL: `http://go.tum.de/623496`.

[26] Stefan Hoch, Frank Reinhold, Bernhard Werner, Kristina Reiss and Jürgen Richter-Gebert. *Bruchrechnen. Bruchzahlen & Bruchteile greifen & begreifen*. Deutsche Version. München: Technische Universität München, 2018. DOI: `10.14459/2018md1436808`.

[27] Stefan Hoch, Frank Reinhold, Bernhard Werner, Kristina Reiss and Jürgen Richter-Gebert. *Fractions. Getting in Touch with Rational Numbers*. English Apple iBooks Version. Munich, Germany: Technical University of Munich, 2018. URL: `http://go.tum.de/423758`.

[28] Stefan Hoch, Frank Reinhold, Bernhard Werner, Jürgen Richter-Gebert and Kristina Reiss. 'Design and research potential of interactive textbooks: the case of fractions'. In: *ZDM Mathematics Education* 50.5 (2018), pp. 839–848. DOI: `10.1007/s11858-018-0971-z`.

[29] Stefan Hoch, Frank Reinhold, Bernhard Werner, Jürgen Richter-Gebert and Kristina Reiss. 'Geschlechtsunterschiede beim Umgang mit dem interaktiven Schulbuch ALICE:Bruchrechnen – eine Analyse von Prozessdaten'. In: *Beiträge zum Mathematikunterricht 2018*. Ed. by Fachgruppe Didaktik der Mathematik der Universität Paderborn. Vol. 4. Münster: WTM-Verlag, 2018, pp. 2075–2076.

[30] Stefan Hoch, Frank Reinhold, Bernhard Werner, Jürgen Richter-Gebert and Kristina Reiss. 'How do students visualize fractions? A finger tracking study'. In: *Proceedings of the 42nd Conference of the International Group for the Psychology of Mathematics Education*. Ed. by Ewa Bergqvist, Magnus Österholm, Carina Granberg and Lovisa Sumpter. Vol. 5. Umeå, Schweden: PME, 2018, p. 64. URL: `https://www.researchgate.net/publication/326122839_How_do_students_visualize_fractions`.

[31] Stefan Hoch, Frank Reinhold, Bernhard Werner, Jürgen Richter-Gebert and Kristina Reiss. *Interaktive Lehrbücher im Bruchrechenunterricht der Sekundarstufe I*. 6. Tagung der Gesellschaft für Empirische Bildungsforschung (GEBF). Basel, 2018.

[32] Daniel Kirsch. 'Detexify: Erkennung handgemalter LaTeX-Symbole'. German. Diploma thesis. Westfälische Wilhelms-Universität Münster, 2010. URL: `http://detexify.kirelabs.org/classify.html`.

[33] Erich Peter Klement, Radko Mesiar and Endre Pap. *Triangular Norms*. Springer, 2000.

[34] Lai-Chong Law, Virpi Roto, Marc Hassenzahl, Arnold Vermeeren and Joke Kort. 'Understanding, scoping and defining user experience: A survey approach'. In: *Proceedings of the 27th International Conference on Human Factors in Computing*. 2009, pp. 719–728.

[35] Yann LeCun, Corinna Cortes and Christopher J.C. Burges. *The MNIST database of handwritten digits*. 1998. URL: `http://yann.lecun.com/exdb/mnist/`.

[36] Scott McCloud. *Understandind Comics – The Invisible Art*. HarperCollins Publishers, 1993.

[37] Hiroyuki Miki. 'Reconsidering the Notion of User Experience for Human-Centered Design'. In: *Human Interface and the Management of Information – Proceedings of the 15th International Conference on Human–Computer Interaction*. 2013, pp. 329–337.

[38] Gordon E. Moore. 'Cramming more components onto integrated circuits'. In: *Electronics* 38.9 (1965).

[39] Mohd Nizam Bin Saad, Hadi Razak and Azman Yasin. 'The Adaptation of Handwriting Recognition System User Interface in Preschool Literacy Learning Courseware'. In: *International Journal of Information and Education Technology* (2012), pp. 61–67.

[40] Friedhelm Padberg and Sebastian Wartha. *Didaktik der Bruchrechnung*. Springer, 2017.

[41] Marc Prensky. 'Digital Natives, Digital Immigrants'. In: *On the Horizon* 9.5 (2001), pp. 1–6.

[42] Janet C. Read, Stuart MacFarlane and Chris Casey. 'Measuring the Usability of Text Input Methods for Children'. In: *People and Computers XV – Interaction without Frontiers*. Springer, 2001, pp. 559–572.

[43] Janet C. Read, Stuart MacFarlane and Peggy Gregory. 'Requirements for the design of a handwriting recognition based writing interface for children'. In: *Proceedings of the 2004 conference on Interaction design and children: building a community*. 2004, pp. 81–87.

[44] Janet C. Read, Stuart MacFarlane and Matthew Horton. 'The Usability of Handwriting Recognition for Writing in the Primary Classroom'. In: *People and Computers XVIII — Design for Life*. 2004, pp. 135–150.

[45] Frank Reinhold. 'Mathematikdidaktische und psychologische Perspektiven zur Wirksamkeit von Tablet-PCs bei der Entwicklung des Bruchzahlbegriffs – Eine empirische Studie in Jahrgangsstufe 6'. German. PhD thesis. Technische Universität München, 2018.

[46] Frank Reinhold, Stefan Hoch, Bernhard Werner, Kristina Reiss and Jürgen Richter-Gebert. *Tablet-PCs im Mathematikunterricht der Klasse 6. Ergebnisse des Forschungsprojektes ALICE:Bruchrechnen*. Münster: Waxmann, 2018. URL: `https://www.waxmann.com/buch3857`.

[47] Frank Reinhold, Stefan Hoch, Bernhard Werner, Jürgen Richter-Gebert and Kristina Reiss. 'iPads in Grade 6 Classrooms: Effects on Students' Choice of Strategy for Comparing Fractions'. In: *Proceedings of the 41st Conference of the International Group for the Psychology of Mathematics Education*. Ed. by Berinderjeet Kaur, Weng Kin Ho, Tin Lam Toh and Ban Heng Choy. Vol. 2. Singapur: PME, 2017, p. 74. URL: `https://www.researchgate.net/publication/318504245`.

[48] Frank Reinhold, Stefan Hoch, Bernhard Werner, Jürgen Richter-Gebert and Kristina Reiss. 'Manipulating Fractions: Effects of iPad-assisted Instruction in Grade 6 Classrooms'. In: *Proceedings of the 41st Conference of the International Group for the Psychology of Mathematics Education*. Ed. by Berinderjeet Kaur, Weng Kin Ho, Tin Lam Toh and Ban Heng Choy. Vol. 4. Singapur: PME, 2017, pp. 97–104. URL: `https://www.researchgate.net/publication/318504242`.

[49] Frank Reinhold, Stefan Hoch, Bernhard Werner, Jürgen Richter-Gebert and Kristina Reiss. 'Konzeptuelles Verständnis von Brüchen mit Visualisierungen auf iPads fördern: Eine empirische Studie'. In: *Beiträge zum Mathematikunterricht 2018*. Ed. by Fachgruppe Didaktik der Mathematik der Universität Paderborn. Vol. 3. Münster: WTM-Verlag, 2018, pp. 1475–1478.

[50] Frank Reinhold, Stefan Hoch, Bernhard Werner, Jürgen Richter-Gebert and Kristina Reiss. *Tablet-PCs im Mathematikunterricht der sechsten Jahrgangsstufe: Das interaktive Schulbuch ALICE:Bruchrechnen*. 109. Bundeskongress des Deutschen Vereins zur Förderung des mathematischen und naturwissenschaftlichen Unterrichts (MNU). München-Garching, 2018.

[51] Frank Reinhold, Sarah Hofer, Stefan Hoch, Bernhard Werner, Jürgen Richter-Gebert and Kristina Reiss. *Schulartspezifische Unterschiede des Einsatzes von iPads im Mathematikunterricht der sechsten Jahrgangsstufe. Ergebnisse einer empirischen Studie und weiterführende Fragestellungen*. Wissenschaftlichen Jahrestagung von LERN 2018 „Digitalisierung und Bildung: Potenziale und Herausforderungen aus der Perspektive der Bildungsforschung". Tübingen, 2018.

[52] Frank Reinhold, Kristina Reiss, Stefan Hoch, Bernhard Werner and Jürgen Richter-Gebert. *Comparing Fractions: The Enactive Way. Supporting Students' Choice of Appropriate Strategies with iPad-Assisted Instruction*. 2018 annual meeting of the American Educational Research Association (AERA). New York, 2018. DOI: 10.302/1303114.

[53] Frank Reinhold, Kristina Reiss, Andreas Obersteiner, Stefan Hoch, Bernhard Werner and Jürgen Richter-Gebert. *Drawing on Children's Intuitive Knowledge to Enhance Fraction Concepts: An Intervention Study with Tablet-PCs*. Fifth meeting of the network „Developing competencies in learners: from ascertaining to intervening". Leuven, Belgien, 2018.

[54] Kristina Reiss, Stefan Hoch, Frank Reinhold, Jürgen Richter-Gebert and Bernhard Werner. 'Tabletklassen: Die Zukunft des Unterrichts?' In: *Bildung im digitalen Zeitalter – Bilanz und Perspektiven*. Ed. by Heinz Nixdorf MuseumsForum. Paderborn: Heinz Nixdorf MuseumsForum, 2017, pp. 92–107.

[55] Jürgen Richter-Gebert. *Perspectives on Projective Geometry*. Springer, 2011.

[56] Jürgen Richter-Gebert and Thorsten Orendt. *Geometriekalküle*. German. Springer, 2009.

[57] Barbara Schmidt-Thieme and Hans-Georg Weigand. 'Medien'. In: *Handbuch der Mathematikdidaktik*. Springer, 2015, pp. 461–490.

[58] Eugene Seneta. *Non-negative Matrices and Markov Chains*. Springer, 1981.

[59] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Prentice Hall, 2001.

[60] Kenneth O. Stanley and Risto Miikkulainen. 'Evolving Neural Networks Through Augmenting Topologies'. In: *Evolutionary Computation* 10.2 (2002), pp. 99–127.

[61] Julie Steele and Noah Iliinsky, eds. *Beautiful Visualisation*. O'Reilly, 2010.

[62] John Sweller. 'Cognitive Load During Problem Solving: Effects on Learning'. In: *Cognitive Science* 12.2 (1988), pp. 257–285.

[63] John Sweller, Paul Ayres and Slava Kalyuga. *Cognitive Load Theory*. Springer, 2011.

[64] Jen Hong Tan and U Rajendra Acharya. 'Active spline model: A shape based model - Interactive segmentation'. In: *Digital Signal Processing* 35 (2014).

[65] Robert T. Taylor, ed. *The computer in school: Tutor, tool, tutee*. Teachers College Press, 1980.

[66] Alessandro Vinciarelli and Juergen Luettin. 'A new normalization technique for cursive handwritten words'. In: *Pattern Recognition Letters* 22.9 (2001), pp. 1043–1050.

[67] Bernhard Werner. *Interactive Widgets*. URL: `https://geo.ma.tum.de/de/personen/bernhard-werner/interaktive-widgets.html`.

[68] Neil L. White. 'Multilinear Cayley Factorization'. In: *International Journal of Information and Education Technology* 11.5–6 (1991), pp. 421–438.

[69] Jin Xiangyu, Liu Wenyin, Sun Jianyong and Zhengxing Sun. 'On-line Graphics Recognition'. In: *Computer Graphics and Applications – Proceedings of 10th Pacific Conference*. IEEE Computer Society, 2002, pp. 256–264.

[70] Kimihiko Yamagishi. 'When a 12.86% mortality is more dangerous than 24.14%: implications for risk communication'. In: *Applied Cognitive Psychology* 11 (1997), pp. 495–506.

[71] Serhiy Yevtushenko. *Concept Explorer*. URL: `http://conexp.sourceforge.net/`.

[72] Cem Yuksel, Scott Schaefer and John Keyser. 'On the Parameterization of Catmull-Rom Curves'. In: *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*. San Francisco, California: ACM, 2009, pp. 47–53.

[73] Poonam Zham, Dinesh K. Kumar, Peter Dabnichki, Sridhar Poosapadi Arjunan and Sanjay Raghav. 'Distinguishing Different Stages of Parkinson's Disease Using Composite Index of Speed and Pen-Pressure of Sketching a Spiral'. In: *Frontiers in Neurology* 8 (2017).