Japanese HWR

Steven B. Poggel
steven.poggel@gmail.com

February 3, 2010

Contents

1 Handwriting Recognition Engine **5**

1.1 Data Capturing . . . 5
1.1.1 Writing Surface . . . 5
1.1.1.1 Writing Surface GUI . . . 5
1.1.1.2 Writing Surfaces Background Module . . . 5
1.2 Data Format . . . 6
1.2.1 Requirements of the Data Format . . . 6
1.2.2 Existing Handwriting Formats . . . 6
1.2.2.1 InkML . . . 6
1.2.2.2 The Microsoft Ink Serialized Format . . . 7
1.2.2.3 The Standard UNIPEN Format . . . 8
1.2.2.4 The UNIPEN Format *UPX* . . . 9
1.2.3 Data Format Description . . . 9
1.2.3.1 Point Data Format . . . 10
1.2.3.2 Stroke Data Format . . . 10
1.2.3.3 Radical Data Formant . . . 10
1.2.3.4 Character Data Format . . . 10
1.3 Database . . . 10
1.4 Recognition Architecture . . . 10
1.5 Stroke Recognition Process . . . 10
1.5.1 Advanced Point Lists . . . 10
1.5.2 Normalisation . . . 10
1.5.2.1 Boxing . . . 10
1.5.2.2 Scaling . . . 10
1.5.3 Curve Handling . . . 10
1.5.4 Dynamic Time Warping . . . 10
1.6 Radical Recognition Process . . . 11
1.7 Character Recognition Process . . . 11
1.8 Error Handling . . . 11
1.8.1 Error Recognition . . . 11
1.8.2 Error Processing . . . 11

Chapter 1

Handwriting Recognition Engine

1.1 Data Capturing

Each handwriting recognition process begins with the data capturing. The user's handwriting must be captured and fed into the system. The data capturing is therefore a crucial part of the whole process. In this system a GUI is used for the capturing of the pen movements on a writing surface.

1.1.1 Writing Surface

The writing surface module, the view is split into two parts. The writing surface GUI and the writing surface background module. The technical design of the data input GUI is described in section **??**.

1.1.1.1 Writing Surface GUI

The GUI works with pen-down and pen-up events. It has a cross in the middle in order to partition the writing surface the same way, character practicing paper sheets are usually partitioned. The GUI class is listening to *pen-down*, *pen-up* and *pen-move* events. These are the equivalent in the mobile world for regular mouse-down, mouse-up and mouse-move events. However, there is one difference - the pen-move event can only be captured between a pen-down and a pen-up event. An earlier conceptual idea for the HWR engine included using the mouse-move events during the input of a character between the strokes. This could not be realised, however, because the series of pen-move events can only be captured when there has been a previous pen-down event and no pen-up event yet. The GUI captures the events and passes the point coordinates on to the background class.

1.1.1.2 Writing Surfaces Background Module

When a pen-down event is detected, the background class of the GUI starts listening for the pen movement. All point coordinates and the time of their capturing are stored in two separate lists with the same indeces. An alternative solution would have been to store a number of instances of a custom-made encapsulated Point class including the time stamp in one list only, however, using two separate lists resulted in increased speed. Therfore, the point coordinates are stored in the frameworks Point class that does not account for time stamps. Therefore a separate list is used for the timestamps only.

The background module of the writing surface mainly administrates the capturing of the pen trajectory and sends it to the recognition module as soon as a stroke is finished. Therefore the system does not receive a separate signal indicating that the drawing of the character is finished. That design creates as segmentation problem that is solved with the *clear* button and the *clear* message. The segmentation of characters is left undetermined - only the beginning of it is determined through the *clear* message that is sent to the moblie view from the controller - or the clear message that is sent to the controller because the user clicked the *reset* button. After a stroke is finished the according point and timestamp sequences are passed to the main part of the HWR engine.

1.2 Data Format

1.2.1 Requirements of the Data Format

The data format for the recognition process underlies a number of requirements. Firstly, it has to be stated that two main data formats are necessary. One for the actual recognition process as a data structure in the RAM. Secondly, there needs be a storage format for the data base that represents the handwriting of a character, radical, stroke, point list or simple point. The requirements for both data structures are:

- **Expressiveness**: The data structure should be able to fully represent a complete character and all sub-elements that belong to it. That requirement is due to the structural approach to handwriting recognition that is the basis for the error handling.
- **Well-definedness**: The information structure should be unambigous, two different characters must have a different structure.

The storage structure has some additional requirements:

- **Accessibility**: The data should be formatted in a way that a human editor can access the data, but it should also be prepared for programmatical access.
- **Well-formedness**: There should be a way to check if the data is well-formed
- **Parsabilty**: It should be possible to create the run-time data structure from the storage data structure.

The run-time data structure should in addition provide

- **Serialisability**: It must be possible to create the storage data structure from the run-time data structure.

Any combination of two formats used should meet the above requirements, at best in a uniform way. That is, any combination of storage format and run-time format should ideally be compatible without the need of a complex data format converter.

1.2.2 Existing Handwriting Formats

There are some existing formats for the description of handwriting. Their intended use are handwriting recognition systems. The formats that will be reviewed here are InkML, NROFF and UPX. It will be analysed if one of these formats meets the above requirements in a sufficient way. This is unclear, because the formats have been developed mainly for alphabetic scripts **??**.

1.2.2.1 InkML

InkML is a markup language for describing electronic pen trajectories. Hence the name *ink*. *ML* stands for markup language, as is customary for XML-based and SGML-based languages. The specification is currently in a draft status, maintained by the World Wide Web Consortium (W3C) (Chee et al. 2006). Judged from the viewpoint of the defined requirements, InkML is a candidate for the storage format. As an XML format it automatically fulfils the criterion of *well-formedness*: Any piece of InkML code can be evaluated with common techniques, since InkML has a well-defined scheme description.

The main elements in the simplest form of InkML are the **<ink>** and the **<trace>** element. These elements allow for a very simple form of a pen trajectory, basically a flat list of point coordinates. The <ink> element is the root element of any InkML code. The <trace> element holds point coordinates.

Listing 1.1: Demonstration of the *trace* tag

```
<ink>
  <trace>
```

```
      282 45, 281 59, 284 73, 285 87, 287 101, 288 115, 290 129,
      291 143, 294 157, 294 171, 294 185, 296 199, 300 213
    </trace>
  </ink>
```

Listing (1.1) covers the general gist of the InkML format. Despite being an XML format, it is a flat format, each value pair, separated by commas, represents one point in a coordinate notation.

It is possible to add *time* information to the trace. In order to do so, a time channel needs to be defined. Listing (1.2) shows the definition and use of a time channel. The example shows a time channel whose values for a given point are the relative to the timestamp refered to to by *#ts001* (Chee et al. 2006). Below there are two *timestamp* tags. The first one has the ID *ts001* and is referred to by both the time channel and the second time stamp that defines a time offset to time stamp *ts001* with the *timeOffset* attribute. The possibility to define time channels enables the InkML format to hold information about the time at which a sample point has been taken. This is useful for the recognition mechanism used in this application, because it compares point coordinates as well as time stamps.

Listing 1.2: Demonstration of the *time channel*

```
<channel name="T"
         type="integer"
         units="ms"
         respectTo="#ts001"/>

<timestamp xml:id="ts001"
           time="1265122414000"/>
<timestamp xml:id="ts002"
           timeOffset="600000"
           timestampRef="#ts001"/>
```

The InkML data structure fulfills several of the requirements necessary for a storage data structure. InkML is

- *well-defined*: Any point sequence that has different coordinate values than another point sequence can be distinguished from the other.
- *accessible*: As an XML format it can be handled programmatically, but is also human-readable and can be edited by a human with a simple interface like a text editor.
- *well-formed*: As an XML format it can be validated according to it's specification.
- *parsabale*: As an XML format, InkML does not need to be parsed with custom-made methods, since most modern high-level languages offer an access method to XML tree structures.

However, InkML is lacking some *expressiveness*. Only pen trajectories and their time stamps can be expressed in the InkML format. InkML is not designed to hold any of the other information compulsory for the structural handwriting recognition. In the case of character recognition, it needs to be able to account for more than coordinate points and time stamps. There needs to be a way to encode structural information about the characters and sub-elements of the characters. InkML does not to be a sufficient format for the given task.

1.2.2.2 The Microsoft Ink Serialized Format

The *Ink Serialized Format* (ISF) format is owned by the Microsoft Corporation. However, it is not a proprietary format, the specification is freely available. Since the format is purely binary, it can not meet one of the requirements for the storage data structure: It is not feasible for a human to read and write files in that format, therefore ISF does not provide the necessary *accessibility* (Microsoft Corporation 2007). The format can still be used as a run-time data structure, if it meets the requested criteria. The requirement

of *serialisability* is generally met. A binary format allows for fast and reliable programmatical access to the data, it can be stored in binary files. The problem that accrues from this type of serialisation is that the storage format, the format used in a file or database, would again not be human-editable and therefore fail to meet the *accessibility* requirement. In order to provide a serialisation that results in a human-readable format, a format converter would be needed. Another problem originates from the ISF format specification: It accounts only for the description of mouse coordinates or pen trajectories, but does not offer any structures for linguistic information. It fails to meet the ideal, a unified format for linguistic and graphic information, as it is lacking some degree of *expressiveness*.

1.2.2.3 The Standard UNIPEN Format

The standard UNIPEN format specification is a file format definition for a flat text file. It contains tags in dot-notation. For example the tag *.COMMENT* means that the following free text should be ignored, the tag *.KEYWORD* is used to define a new keyword, while the tag *RESERVED* states that the text that comes after that tag is a reserved word within the UNIPEN format. The format is self-defined with the three keywords above. Any new keyword is defined with *.KEYWORD* (Guyon et al. 1994; Unipen Foundation, International 2010).

As a data structure for the purpose of a handwriting recognition, UNIPEN can serve as a storage format, which is the primary purpose for the existence of the format. The UNIPEN format accounts for both pen trajectories as well as information about the characters that have been written.

Listing 1.3: Demonstration of the *UNIPEN* format

```
.COMMENT ##################################################################
.COORD   X Y

.SEGMENT TEXT 235:0-297:9 OK "Kurosu Masaaki"
.SEGMENT CHARACTER 235:0-255:9 OK  "JISx3975 'Kuro'"

.PEN_DOWN
  486 -1456
  488 -1454
  490 -1452
  488 -1450
  488 -1450
  486 -1452
  480 -1456
  474 -1466
  464 -1480
  452 -1492
  440 -1506
  428 -1524
.PEN_UP
  406 -1556
  394 -1574
  384 -1590
  374 -1602
.PEN_DOWN
```

Listing (1.3) shows a short example for a UNIPEN data structure. The *.COORD* tag defines the structure of the pen coordinates. The *.SEGMENT* tag with the *TEXT* modifier informs about the full text of the next number of segments. The *.SEGMENT* tag with the *CHARACTER* modifier informs about the character that is represented with the sequence of pen coordinates. The other information given is the start and end position within the file and the character code in JIS encoding. The *.PEN_DOWN* tag can be understood

as a flag. All pen coordinates following this tag have been captured as *pen down* coordinates. The tag *.PEN_UP* sets the opposite flag: The coordinates stated after this tag were captured while the pen was not touching the writing surface. The last *.PEN_DOWN* tag is the beginning of a new coordinate sequence for the next stroke. From a requirements point of view the UNIPEN provides a high standard, as it meets a great number of requirements for the storage structure.
The UNIPEN format is:

- partially **expressive**: UNIPEN can represent a complete character and the pen trajectory along with it. It does however, not account for capturing the time information. At a constant sample rate the time stamps of the individual points could theoratically be calculated, if time stamp of the first point is encoded in the meta information. However, that solution is not optimal, because it needs constant sampling, even between a *pen-up* and a *pen-down* event. There are input devices that do not offer those coordinates and the recognition system presented here does not rely on them for that reason. The examples given are encoded in JIS, but it seems possible and feasible to use the format with unicode encoding.

- **well-defined**: The UNIPEN is unambigous, two different characters do have a different data structure, because the JIS code can serve as an ID.

- **accessibile**: The data resides in flat text files, designed for human editing. It is accessble programmatically, too, but has a weakness on that requirement compared to an XML-based format.

- **parsable**: The flat file format can be parsed easily. Due to its procedural nature it is also possible to create a run-time data structure that is conceptually based on the storage structure and therefore serialisable.

The main weakness of the flat format file is the lack of *well-formedness*. There is no automatic way to check a document against a predefined format specification. It can not be assessed if a file is well-formed. Additionally, the expressive power of the UNIPEN format would be seriously challenged if it should provide for sub-structures of characters. In order to do so, it would be necessary to create new tag definitions for the Radicals of the Kanji. Therefore, it can be concluded that the standard UNIPEN format is not suitable for the task given.

1.2.2.4 The UNIPEN Format *UPX*

(Agrawal et al. 2005; Unipen Foundation, International 2010)

1.2.3 Data Format Description

A general description of the XML format - why it was used that way, given the requirements

data format of and representation of point, stroke, box, radical, character s. 20-23, 25f

show process of how I came to current data formats. include character models 1-4 on pages. - however, not everything in one go, but rather in the individual sections if possible. in the end, a radical has its own format that is unchanged, even if internal structure of a stroke is changed. (unipen is only text based, inkml does not help here, but the system allows for exchange of the custom format with those)

1.2.3.1 Point Data Format

1.2.3.2 Stroke Data Format

1.2.3.3 Radical Data Formant

1.2.3.4 Character Data Format

1.3 Database

1.4 Recognition Architecture

1.5 Stroke Recognition Process

1.5.1 Advanced Point Lists

what happens to the points? nothing, really - the magic happens when normalisation and the other stuff starts. why this section? what's the purpose? oh, right - angles and vectors instead of simple points. from one point to the next, or rather from on point to ten points down the line, to get a rougher direction. vectors make it interesting. impacts on curve handling! gradient and stuff can be measured in the vector representation (even without any boxes) making the point list a cool mathematical object! show code samples in pseudocode if necessary. report about the cool stuff.

what's the similarity measure for points and strokes? show requirements. what alternatives were there to consider?

1.5.2 Normalisation

what is N? why do N? show requirements. how is N performed here? why is it performed like that?

1.5.2.1 Boxing

how is boxing done? show requirements. what alternatives were there to consider? is it useful to have a similarity measure for bounding boxes? yes! but why? explain! size of the boxes! - think of characters that only have two strokes.

1.5.2.2 Scaling

s. 42-45 how is scaling done? show requirements. what alternatives were there to consider?

1.5.3 Curve Handling

S 14, 16, 17 how is curver handling done? show requirements. what alternatives were there to consider?
stroke matching with angles instead of point position. s. 24

1.5.4 Dynamic Time Warping

what's the similarity measure for points and strokes? show requirements. what alternatives were there to consider?

s. 51 how is dynamic time warping done here? pointer to papers or hwr - chapter, don't explain DTW here. show requirements why DTW? what alternatives were there to consider? none - it is the alternative. to all the other stuff I've been doing. however, what about 3D time warping?

1.6 Radical Recognition Process

1.7 Character Recognition Process

1.8 Error Handling

see section **??** in chapter **??** for possible sources of error

1.8.1 Error Recognition

why this section? to demonstrate own achievements of error recognition. the reader should know how it is done technically.

what goes into this section? the aspects of finding errors. finding errors is not a straightforward trivial task - whenever something does not match it is an error - doesn't work like that. instead, firstly, it needs to be made sure that it actually is an error. meaning - not a recognition error, but a user error. secondly, the type of error needs be identified. see section **??** (or handwritten page 58) for sources of error.

how will this section be written? technical - first describe how the error recognition integrates into the recognition process, then how errors are identified.

1.8.2 Error Processing

why this section? actually the 'handling' or 'processing' aspect could be described in the recognition section 1.8.1 as well. so this section is only for a better overview, for document structure, thematically they are the same section. thus they are put together under Error Handling 1.8.

what goes into this section?

List of Figures

Listings

1.1 Demonstration of the *trace* tag 6
1.2 Demonstration of the *time channel* 7
1.3 Demonstration of the *UNIPEN* format 8

List of Tables

References

Agrawal, M., K. Bali, S. Madhvanath, and L. Vuurpijl (2005). UPX: A new XML Representation for Annotated Datasets of Online Handwriting Data. In *ICDAR*, pp. 1161--1165.

Chee, Y.-M., K. Franke, M. Froumentin, S. Madhvanath, J.-A. Magaña, G. Russell, G. Seni, C. Tremblay, S. M. Watt, and L. Yaeger (2006, Oct). *Ink Markup Language (InkML)*. W3C Consortium. Online. Retrieved from `http://www.w3.org/TR/2006/WD-InkML-20061023/` on 2010-02-01.

Guyon, I., L. Schomaker, R. Plamondon, M. Liberman, and S. Janet (1994). UNIPEN project of on-line data exchange and recognition benchmarks. In *Proc. of the 12th International Conference on Pattern Recognition*, Jerusalem, pp. 29--33.

Microsoft Corporation (2007). *Ink Serialized Format Specification*. Microsoft Corporation. Online. Retrieved from `http://download.microsoft.com/download/0/B/E/0BE8BDD7-E5E8-422A-ABFD-4342ED7AD886/InkSerializedFormat(ISF)Specification.pdf/` on 2010-02-02.

Unipen Foundation, International (1994-2010). Unipen Specification. Online. Retrieved from `http://www.unipen.org/]` on 2009-10-25.