German Lute Tablature Recognition

Christoph Dalitz Christine Pranzas
Niederrhein University of Applied Sciences
Reinarzstr. 49, 47805 Krefeld, Germany
christoph.dalitz(at)hsnr.de, christine(at)pranzas.com

Abstract

This paper describes a document recognition system for 16th century German staffless lute tablature notation. We present methods for page layout analysis, symbol recognition and symbol layout analysis and report error rates for these methods on a variety of historic prints. Page layout analysis is based on horizontal separator lines, which may interfere with other symbols. The proposed algorithm for their detection and removal is also applicable to other single staff line detection problems (like percussion notation), for which common staff line removal algorithms fail.

Table 1. Symbol meaning in German tablature up to the sixth position ("fret"). "Course 1" means the highest string.

fret no:	0	1	2	3	4	5	6
course 1	5	e	k	p	v	9	ē
course 2	4	d	i	o	t	⁊	d̄
course 3	3	c	h	n	s	ʒ	c̄
course 4	2	b	g	m	r	y	b̄
course 5	1	a	f	l	q	x	ā
(*course 6*)	𝔄	𝔅	ℭ	𝔇	𝔈	𝔉	𝔊

1. Introduction

During the 16th century, a great number of music notations for string instruments were in use. For lute alone, more than four different notational systems were used, known as "Italian", "French", "Spanish" and "German" tablature based on their predominant regional use [1]. Of these, only two systems are still in general use today: "Spanish" tablature has developed into the modern guitar tablature, and "French" tablature is the tablature system preferred by lutanists today. The peculiarity of German lute tablature is that it does not use staff lines to identify the courses of the instrument, but uses a staffless notational system with symbols semantically different from the other notations.

There is a large body of historic tablature prints and manuscripts preserved; a fairly complete catalog of extant sources can be found in [2]. These provide an interesting application area for document recognition techniques. Actually, one of the authors has already developed a recognition system for tablature system utilizing staff lines [3]. Another recognition system was recently developed independently by Wei et al. for modern guitar tablature [4]. Both of the existing systems use the staff lines for page layout analysis. The system by Wei et al. additionally assumes that the tablature symbols be only numbers (0-9), and it relies on a strictly proportional horizontal spacing of chords, an assumption that does not hold for historic prints. So neither of the existing OTR systems is applicable to the staffless German tablature, thus making new approaches and experiments necessary which are described in the present paper.

We have implemented our recognition system as a "toolkit" for the *Gamera* framework for document image analysis[1]. Gamera is a cross platform Python library for building custom recognition systems that has already been used successfully for the recognition of various non standard document types like documents in the Navajo language or Byzantine chant notation [5]. Like Gamera itself, we make our source code freely available under the GNU general public license[2].

2. German lute tablature

Tablature is a musical notation for string instruments that does not specify the sound of the music, but rather when and in which positions ("frets") the strings ("courses") of the instrument are stopped. Most tablature systems use staff lines to indicate the course and numbers or letters to specify the fret, such that the same fret character can appear on different lines. German tablature in contrast does not use staff

[1]see http://gamera.sourceforge.net/
[2]see http://otr4gamera.sourceforge.net/

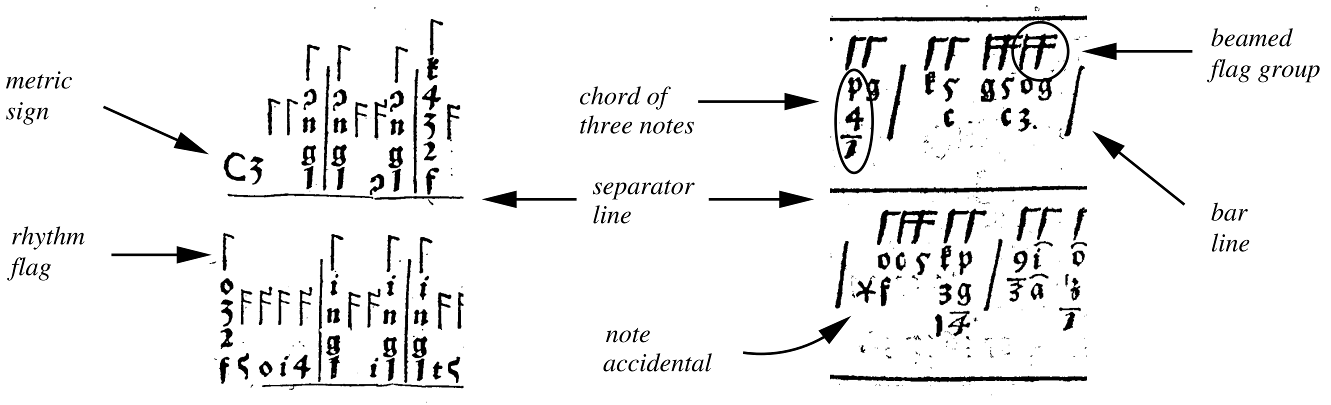

Figure 1. Common properties of two historic German lute tablature prints: Judenkünig 1532 (left) and Gerle 1552 (right).

lines, but uses unique characters for specifying *both* course and fret as shown in Tbl. 1.

When the alphabet runs out after the fifth course, the characters are periodically reused by placing a stroke above. As the sixth course was added to the instrument after the original invention of German lute tablature, different crutches were used for its frets like upper case letters, stroked through numbers, numbers with a stroke above or even mixtures of these.

In the following, we will call the symbols from Tbl. 1 "notes" because they uniquely specify pitches (assuming a fixed tuning of the instrument), like the notes in common music notation. Fig. 1 shows excerpts of two typical historic prints. While the individual shapes for the "note" characters can be different, there are a number of common general properties:

- The tablature is organized in lines which are separated by wide horizontal lines.
- Notes that are plucked simultaneously are vertically aligned to form "chords".
- Rhythm flags specifying the duration to the next chord appear above the notes. The flags can be beamed similar to modern music notation resulting in a finite set of grid shaped flag groups.

While the symbol variety between different prints is best treated with a training based statistical classifier, these common features are to be utilized for page segmentation and symbol layout analysis.

3. The recognition system

The recognition process in our system consists of the four steps *preprocessing*, *segmentation*, *classification* and *symbol layout analysis* which are performed sequentially. These steps are described in detail in the following subsections.

The output of our system is an ASCII encoding of the tablature in Dalitz' extension of the *abc* format[3]. Due to the current lack of a widely accepted lute tablature encoding standard we have used this format because of the availability of commodity software. There is however nothing specific about this code, and the system can easily be adjusted to different output formats like Crawford's *TabCode* [6].

3.1. Preprocessing

The images were scanned with a resolution of 300 dpi and binarized with Otsu's global threshold. To improve the quality of the resulting onebit images, we have removed black borders ("copy margins") by flood filling from black border pixels, corrected a possible rotation with the projection profile based method described in [5] and removed black and white speckles as connected components containing less than eight pixels on the original and inverted (for white speckles) image. Remaining speckles were left to be sorted out at the classification stage.

Some prints (e.g. Waissel 1573 in Tbl. 3) showed highly fragmented symbols due to poor printing quality. For these prints, we additionally did a morphological closing operation with a 3×3 structuring element. This somewhat remedied the fragmentation, but introduced other problems (see Sec. 4).

3.2. Page segmentation

The goal of the page segmentation step is to segment the image into lines of tablature and to remove the sepa-

[3]see `http://www.lautengesellschaft.de/cdmm/`

rator lines. Most human readers probably use the texture of the rhythm flags to identify tablature lines. This approach however requires knowledge about the symbol shapes in a particular print and is thus classification based and not image independent. We therefore utilized instead the separator lines for page segmentation, which must be found and removed anyway, as they often touch the symbols.

Formally, the problem of finding and removing the separator lines is similar to the problem of staff line removal in common music notation. None of the algorithms described in [7] is however applicable to *single* staff line removal due to their reliance on an estimator "*staffspace_height*" for the distance between adjacent lines within the same staff group. We therefore devised our own algorithm, based on Carter's idea of analyzing the adjacency of black vertical runlengths [8].

This approach requires an estimator for the line height of the separator lines. To be able to estimate this value from the maximum in the histogram of black vertical runlengths [7], we followed the idea in [5] to filter out some connected components, so that the runlength histogram of the remaining image is dominated by the separator lines. We estimate two dimensions *character_height* (the height of a "note" character) and *line_height* (the height of a separator line) as follows:

- *character_height* is set to the median height of all connected components (CCs)
- to avoid that separator lines form a single CC with touching vertical lines (from bar lines or a surrounding frames), we remove all vertical black runlengths greater than *character_height*
- of the remaining CCs, all CCs with an aspect ratio $width/height < 3$ are removed so that only wide CCs remain, of which most are horizontal lines
- on the resulting filtered image, we set *line_height* to the most frequent black vertical runlength; other than for the estimation of *line_height*, the filtered image is not used

Our algorithm for finding the separator lines first collects line segment candidates as filaments of adjacent black vertical runs shorter than $2*$*line_height*, a threshold that breaks the lines at crossing symbols. These filaments are then divided into those wider than 10% of the total image width and those narrower. The wide filaments are further partitioned into equivalence classes of filaments, where the equivalence inducing criterion is a vertical distance less than $6 *$ *line_height*, a threshold that allows for some curvature and vertical offsets within a line. Each of the resulting filament sets forms a horizontal line, but not necessarily a separator line.

To pick the separator lines from these "horizontal lines", they are again partitioned into equivalence classes, where now the equivalence inducing criterion is a vertical distance less than $3*$*character_height*, a distance that corresponds to a two part chord with a rhythm flag. In each resulting class, we pick the "horizontal line" containing the most vertical runlengths as a separator line. From the separator lines, we estimate the tablature *system_height* as the median distance between adjacent separator lines. As separator lines are often broken in the historic print, the lines found so far are only approximate, and occasionally even a complete line might have been missed.

We have therefore added two final steps to the line finding algorithm, taking *all* line filaments (wide and narrow) into account. By adding filaments close to the least square fitted separator line, gaps are filled and filaments lying too far away are replaced with closer filaments in the line. Moreover, missed separator lines are detected by looking for filaments around the middle between two lines that are more than $1.5 *$ *system_height* apart, a distance unlikely to occur when all systems are of similar height as was the case in all examined prints.

As the found lines consist only of the line segments without possibly superimposing objects, none of the sophisticated methods described in [7] is necessary to separate line segments from symbols. Removing all line segments thus only removes the lines while keeping touching or crossing symbols intact.

3.3. Symbol classification

Once the separator lines are found and removed, the individual symbols are isolated by connected component labeling and then classified with a nearest neighbor (NN) classifier. For classification, we have chosen the feature set *aspect_ratio, moments, volume64regions, nrows*[4], because previous experiments on different prints have shown that from all of Gamera's builtin features, this combination had the lowest error rate, both with respect to a holdout error estimate [3] and a cross correlation error estimate [5].

In addition to the classes representing actually meaningful glyphs, we have introduced a class "trash" for noise. This was necessary because even after preprocessing the images were still considerably noisy and we have let the classifier sort out the noise. As noise can have any shape, but is typically small, our feature set included the scale variant feature *nrows*, which is simply the height of a glyph.

To deal with the problem of broken symbols, we have used Droettboom's classification based grouping algorithm [9]. To deal with the problem of touching rhythm flags forming beamed groups (see Fig. 1 right), we have trained beamed groups as a whole using a special class naming convention for the number of stems and rhythm values within the group, e.g. `flag.2.1.4.2` for , i.e. note value

[4]see the Gamera documentation for their precise definition

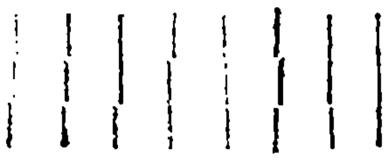
Figure 2. Examples for bar lines in Waissel 1573 (after preprocessing with closing)

1/2 once followed by note value 1/4 twice. This was possible, because, unlike in common music notation, the beams in German lute tablature do not vary in slant direction, but are of rather fixed shapes.

While the NN classifier turned out to be quite reliable for notes and rhythm flags, the same did not hold for bar lines. One reason for this problem was that flagless rhythm stems (representing a whole note) are identical to bar lines. Another, more fundamental problem were broken bar lines in which the relative position of the parts varied (see Fig. 2). These made it necessary to set the parameters in Droettboom's grouping algorithm [9] for the maximum number of parts per group and their distance so high, that the algorithm became both slow and error prone, due to the necessity to check too many possible subgraphs.

We therefore devised two additional alternative classification rules for bar lines. One consists in training and classifying bar line fragments as *bar_part* and then joining adjacent *bar_parts* with a small horizontal distance. The other approach does not include training and is similar to the separator line detection described in Sec. 3.2, but analyzes the adjacency of *horizontal* black runlengths rather than *vertical* runlengths. Our experimental results described in Sec. 4.2 show that for prints with fragmented bar lines, the purely rule based approach outperforms the other two.

3.4. Symbol layout analysis

Once the symbols are classified, they need to be organized as a linear sequence of chords with rhythm values attached. In the earlier recognition system for French and Italian lute tablature [3], this grouping was based solely on building equivalence classes of glyphs based on horizontal overlap. In the case of German lute tablature, this is however not sufficient, because rhythm flags can be beamed and thus span several chords. Moreover, the individual chords are often printed so close to each other that they overlap significantly (see Fig. 3).

To deal with beamed flags, we have split the chord grouping into two steps. First the notes are grouped according to their horizontal overlap. Then the note chords are attached to the rhythm flag with which they have the largest horizontal overlap, with the additional constraint that

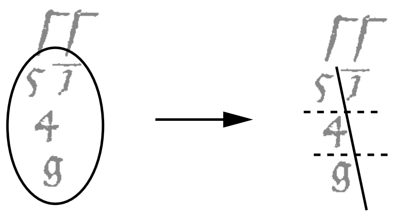
Figure 3. Close by note chords can overlap and need further splitting (Gerle 1552)

no rhythm flag should have more note chords than stems as specified in its class name (see Sec. 3.3). Overlapping note chords can be detected from different symbols at a similar y-position within the same chord. These are segmented by matching the chord to a grid of rows and columns.

4. Results

We have tested our recognition system on facsimile reprints of different 16th century German lute tablature prints. These facsimile prints were particularly suited for testing the system, because they reproduce image defects of the original prints, which the publishers have not tried to remedy. We have counted the error rates both for the separator line detection algorithm and the symbol recognition and grouping.

4.1. Separator line detection

From nine different prints, we have picked a random set of 20 pages each, resulting in totally 1263 separator lines on 180 pages. Of these, only three lines were missed and one line was falsely found, which is an error rate of about 0.3%. The four errors occurred in only two prints, which had significant show through (Judenkünig 1532) or highly fragmented lines and characters (Ochsenkuhn 1558). The fragmentation can have the effect, that not sufficiently many "wide" filaments are found in the first stage of our line detection algorithm. Nevertheless, even on this print only 3 out of 206 lines were not correctly detected.

4.2. Symbol recognition and grouping

As described in Sec. 3.3, the recognition of bar lines posed more problems than the recognition of the other symbols. We therefore evaluated our three approaches to bar line recognition on 140 pages of seven prints. The results are shown in Tbl. 2, where *Gamera* stands for NN with Droettboom's grouping, *bar_parts* for classifying fragments and *runlength* for the rule based approach utilizing a runlength analysis. It turned out that on prints without broken

historic print	*bar lines*	*error rate (%)* runlength	*Gamera*	*bar_parts*
Gerle 1546	459	1.74	**0.21**	3.49
Gerle 1552	456	1.97	**1.32**	4.82
Judenkünig 1523	937	**1.71**	13.98	4.82
Newsidler 1536	422	**3.55**	45.02	45.02
Newsidler 1544	362	**0.82**	22.38	3.31
Newsidler 1549	452	**3.98**	27.21	32.08
Waissel 1573	823	**6.31**	49.21	31.59

Table 2. Bar line recognition error rates for our three different approaches. Bold face indicates the lowest error.

historic print	*symbols*	*error rate (%)*
Gerle 1546	1881	2.2 ± 0.7
Gerle 1552	2161	6.7 ± 1.1
Judenkünig 1523	1995	4.5 ± 0.9
Newsidler 1536	2352	2.8 ± 0.7
Newsidler 1544	1794	3.4 ± 0.8
Newsidler 1549	1825	6.4 ± 1.1
Waissel 1573	3692	10.8 ± 1.0

Table 3. Error rates of our recognition system on historic prints (see [2] about the sources).

bar lines (the two books by Gerle) the NN classifier was best, while on books with broken bars, the purely rule based approach performed best.

The performance of the entire system on these prints is shown in Tbl. 3. For each print, the system has been trained on two pages and then tested on six pages. The given error rate is the number of wrong, unmatched or missing output symbols divided by the total number of symbols. The confidence intervals are Agresti-Coull intervals [5] for a confidence level $\alpha = 0.05$. It should be noted that the reported error rates not only include errors due to misclassifications, but also layout analysis errors.

From Tbl. 3, we conclude that the recognition rate of our system strongly depends on the print quality. The two Newsidler prints from 1536 and 1549 for instance use the same typeface, but the later print shows more printing defects and noise. The poor performance on Waissel's print from 1573 was due to severely broken characters. Some of these defects were somewhat remedied by morphological closing during preprocessing, which however introduced new problems by occasionally joining different characters.

5. Conclusions

The present work demonstrates that the Gamera framework provides a solid foundation for building a custom recognition system for German lute tablature. For prints of decent quality, the recognition rates of our system are quite good. In case of highly fragmented glyphs however, our CC based glyph segmentation approach causes problems. Even though Gamera has a builtin algorithm for dealing with broken characters, this turned out to only yield limited results in our experiments. We therefore consider further research concerning broken character recognition to be crucial for improving the recognition of historic tablature prints.

Acknowledgements

We thank the Cornetto Verlag (Stuttgart) for the kind permission to reproduce details from its facsimile reprints of Gerle 1552 and Waissel 1573, and Dr. Bernd Christoph Becker (Cologne) for the kind permission to reproduce details from his facsimile reprint of Judenkünig 1532.

References

[1] D.A. Smith: "A History of the Lute from Antiquity to the Renaissance." The Lute Society of America, 2002

[2] E. Pohlmann: "Laute, Theorbe, Chitarrone." Eres Edition, Bremen, 1982

[3] C. Dalitz, T. Karsten: "Using the Gamera Framework for building a Lute Tablature Recognition System." Proceedings ISMIR, pp. 478-481, 2005

[4] L.L. Wei, Q.A. Salih, H.S. Hock: "Optical Tablature Recognition (OTR) System: Using Fourier Descriptors as a Recognition Tool." Proceedings ICALIP, pp. 1532-1539, 2008

[5] C. Dalitz, G.K. Michalakis, C. Pranzas: "Optical Recognition of Psaltic Byzantine Chant Notation." International Journal of Document Analysis and Recognition 11, pp. 143-158, 2008

[6] G. Wiggins, T. Crawford, M. Gale, D. Lewis: "An Electronic Corpus of Lute Music (ECOLM)". `http://www.ecolm.org/`, 1999-2006

[7] C. Dalitz, M. Droettboom, B. Pranzas, I. Fujinaga: "A Comparative Study of Staff Removal Algorithms." IEEE Transactions on Pattern Analysis and Machine Intelligence 30, pp. 753-766, 2008

[8] N.P. Carter, R.A. Bacon: "Automatic Recognition of Printed Music." In H.S. Baird, H. Bunke, K. Yamamoto (editors): "Structured Document Image Analysis", pp. 454-65, Springer, 1992

[9] M. Droettboom: "Correcting broken characters in the recognition of historical printed documents." Joint Conference on Digital Libraries, pp. 364-366, 2003