AUTOMATIC LINE SEGMENTATION
IN LATE MEDIEVAL LATIN MANUSCRIPTS
A THESIS SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE
MASTER OF SCIENCE
BY
NICOLAS P. RENET
(ADVISER: DR. J. MICHAEL MCGREW)
BALL STATE UNIVERSITY
MUNCIE, INDIANA
JULY 2012
ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Michael McGrew and Judy Bonneau for their encouragement and their patience. Special thanks go to my family: to Anne-Marie and Bernard Renet, Caroline, Katherine, and Richard Thorington who provided me with the gift of time to finish this thesis; to Etienne and Matthieu, who acted with grace when the storyteller was missing at bedtime; and finally to Ellen Thorington, who provided all that and much more: only she knows how much I am indebted to her loving trust.
CONTENTS

ACKNOWLEDGEMENTS ......................................................................................... ii

LIST OF TABLES .................................................................................................. v

LIST OF FIGURES ................................................................................................. vi

1. Line segmentation techniques in off-line handwriting recognition ............... 1

  1.1 Statement of the problem ............................................................................ 1

   1.1.1 Historical handwritten documents: a challenge for the OCR field ........ 1

   1.1.2 Off-line vs. on-line recognition ........................................................... 3

   1.1.3 Line segmentation in off-line OCR treatments ................................... 4

  1.2 Definitions ................................................................................................. 7

  1.3 Related work ............................................................................................. 8

   1.3.1 Projection-based methods ................................................................... 9

   1.3.2 Smearing methods .............................................................................. 11

   1.3.3 Grouping methods ............................................................................. 12

   1.3.4 Hough Transform .............................................................................. 14

   1.3.5 Mixed methods, other methods .......................................................... 14

   1.3.6 Remarks ............................................................................................. 15

      1.3.6.1 Dependence on user-defined parameters .................................... 16

      1.3.6.2 Extracting metrics ....................................................................... 16

      1.3.6.3 The role of the connected component analysis ......................... 17

      1.3.6.4 Dealing with touching characters ............................................. 18

  1.4 Scope of the current work .......................................................................... 19

   1.4.1 Sample documents: "hard" manuscripts from the 15th century ............... 19

   1.4.2 Constraints ......................................................................................... 20

2. Line segmentation with the profile method ..................................................... 23

  2.1 Binarization problems .............................................................................. 23

   2.1.1 Properties of the document images ................................................... 23

      2.1.1.1 Color coding ......................................................................... 23

      2.1.1.2 The manuscript’s palette ......................................................... 24

      2.1.1.3 Paths to binarization ............................................................... 25
2.1.2 Working with a grayscale document ........................................... 26
  2.1.2.1 Global thresholding (Otsu) .............................................. 26
  2.1.2.2 Adaptive binarization (Sauvola) ....................................... 28
2.1.3 Working with the color image .................................................. 31
  2.1.3.1 Color reduction ........................................................... 31
  2.1.3.2 Production of a binary image ........................................... 33
2.2 Profile smoothing ................................................................. 34
  2.2.1 Local linear kernel smoothing .............................................. 34
  2.2.2 Kernel smoothing is not easily automated ................................. 37
  2.2.3 Profile-based estimate of the text block boundaries ................... 38
2.3 Estimating the interlinear spacing by Fourier analysis ................. 39

3. Feature-based line segmentation ................................................. 43
  3.1 An overview ................................................................. 43
  3.2 Describing the handwritten material ........................................ 44
    3.2.1 Connected component analysis ......................................... 44
      3.2.1.1 Component labeling ................................................. 44
      3.2.1.2 Using the CC analysis .............................................. 46
    3.2.2 Thinning ................................................................. 48
      3.2.2.1 Thinning preserves the pattern topology .......................... 48
      3.2.2.2 A 4-pass thinning algorithm ...................................... 51
      3.2.2.3 A 2-pass thinning algorithm ...................................... 56
      3.2.2.4 Complexity .......................................................... 58
      3.2.2.5 Two other algorithms .............................................. 61
    3.2.3 Identifying segments and nodes ........................................ 67
      3.2.3.1 Nodes ................................................................. 67
      3.2.3.2 Segments ............................................................ 68
  3.3 Clustering-based segmentation ................................................. 70
    3.3.1 Clustering the nodes .................................................... 71
    3.3.2 Assigning the segments to the clusters ................................ 79
      3.3.2.1 Bridge-segments .................................................... 79
      3.3.2.2 Assignment heuristics for bridge-segments ...................... 82
      3.3.2.3 Assigning free segments ........................................... 83
4. Tests and conclusions ....................................................... 88
  4.1 Sample manuscripts .................................................... 88
  4.2 Test results ............................................................... 90
    4.2.1 Scoring system ..................................................... 90
    4.2.2 Successful performance of the algorithm ...................... 92
    4.2.3 An example of algorithm failure ............................... 92
  4.3 Possible improvements ............................................... 96
    4.3.1 Residual errors .................................................. 96
    4.3.2 Genericity ......................................................... 96
    4.3.3 Restoring the character strokes from the thinned segments 98
  4.4 General conclusions and further researches ....................... 99

Bibliography ................................................................. 100

A. Ms. HRC leaf M3, folio 96v ........................................... 106
  A.1 Ms HRC leaf M3, folio 96, recto ................................. 107
  A.2 Ms HRC leaf M3, folio 96, recto: left column ................ 108
  A.3 Ms HRC leaf M3, f96r: binarized .............................. 109
  A.4 Ms HRC leaf M3, f96r: connected components map ............ 110
  A.5 Thinning with BM2+4-pass procedure ........................... 111
  A.6 Segmented manuscript ............................................. 112

B. Test manuscripts ....................................................... 113


D. Implementation .......................................................... 127
  D.1 A simplified view ................................................... 128
  D.2 The detailed view ................................................... 130
LIST OF TABLES

4.1 Testing set ................................................................. 89
4.2 Recall statistics ......................................................... 91
### LIST OF FIGURES

1.1 Book of Hours (MS. UCB 130, Gothic script, France, XV) .......................... 1
1.2 Cursiva textualis script, Germany, XV ................................................. 2
1.3 Latin vocabulary (Noting hand, Italy, XV) .............................................. 3
1.4 Latin vocabulary (Noting hand, Italy, XV) - abbreviation marks ................ 3
1.5 The off-line OCR pipe-line ....................................................................... 5
1.6 Describing the handwritten line ............................................................ 7
1.7 Vertical pixel value projection (MS. UCB 130, France, XV) ...................... 9
1.8 Ms. HRC153 with blurred lines .............................................................. 11
1.9 Centroids arrangement through tree-based location search (Gupta, 2010) ... 13
1.10 Segmentation of an Arabic document through block covering .................. 15
1.11 Cut pixel minimization path ................................................................. 18
1.12 Local touching path (Kang and Doermann, 2011) ..................................... 19
2.1 Color histograms .................................................................................... 24
2.2 Ms. HRC leaf M3: 8 color reduction through median cut quantization ........ 25
2.3 Binarization of color documents ............................................................. 26
2.4 Otsu automatic thresholding operator ..................................................... 28
2.5 Choosing the parameters for Sauvola’s adaptive thresholding ................... 29
2.6 Ms. HRC leaf M3 f96r: horizontal black run lengths ................................. 30
2.7 Reduction to 3-bit colorspace (8 colors) .................................................. 32
2.8 Reduction to 3-bit colorspace (8 colors) .................................................. 32
2.9 Reduction to 3-bit colorspace (8 colors), from a single threshold .............. 33
2.10 Ms. HRC leaf M3, f96r: binarized image ............................................... 34
2.11 Vertical profile of Ms. HRC_leaf_M3 (rotated) ........................................ 35
2.12 Smoothing a profile by local linear kernel regression .................................. 36
2.13 Obtaining line metrics with a thresholded profile ........................................ 37
2.14 Profile-based line detection in Ms. FO.Z113.P3 ............................................. 38
2.15 Block boundaries computation ................................................................. 40
2.16 Ms. HRC leaf M3, f86r: Fourier power spectrum of the vertical ink profile. ... 41
3.1 Feature-based segmentation: overview ......................................................... 45
3.2 Dual adjacency model .................................................................................. 46
3.3 Connected components map (Ms. HRC leaf M3, f96r) .................................. 47
3.4 Connected components statistics ................................................................. 48
3.5 Thinning: obtaining centrally located arcs through pixel erosion ................... 49
3.6 Thinning as a 1-to-1 mapping between connected components (Ms HRC leaf
3, f86r - detail) ........................................................................................................ 50
3.7 Two types of border pixels ........................................................................... 53
3.8 A few 3 × 3 contexts for non-deletable pixels ............................................. 53
3.9 3 × 3 masks for North deletable border pixels (deletion step 1) .................... 54
3.10 Corner patterns common to deletion steps 1 (N) and 2 (E) ............................ 54
3.11 4-pass thinning algorithm: Ms HRC leaf M3, f86r (detail) ........................... 55
3.12 4-adjacent pixels pattern and its rotations .................................................. 56
3.13 4-pass thinning algorithm: difficult cases ..................................................... 57
3.14 Noise in thinned characters ......................................................................... 57
3.15 2-pass thinning algorithm: Ms HRC leaf M3, f86r (detail) ........................... 59
3.16 2-pass thinning algorithm: difficult cases ..................................................... 59
3.17 Eroding a square .......................................................................................... 61
3.18 Thinning and restoring masks for Chin and Wang’s algorithm ...................... 62
3.19 Delete and restore masks (Chin & Wan’s algorithm, 1987) ........................... 63
3.20 Chin & Wan’s thinning algorithm (1987) ...................................................... 64
3.21 Thinning and restoring masks for Bernard and Manzanera’s algorithm ...... 64
3.22 BM2 thinning algorithm (1999) ................................. 65
3.23 Skeletons: comparison .............................................. 66
3.24 Segments and nodes ............................................... 67
3.25 Node detection patterns ........................................... 68
3.26 Manuscript HRC_leaf_M3, f96r: Nodes ...................... 69
3.27 Segment construction: pixel search order ...................... 70
3.28 Manuscript HRC_leaf_M3, f96r: Segments ..................... 71
3.29 Overview of the segmentation procedure: node clusters and bridge-segments 72
3.30 An iteration of the \( k \)-means clustering process (from Manning & Schütze, 1999) 73
3.31 Recomputing a cluster centerline with a least-square fit ............... 73
3.32 Assigning nodes to North and South half-clusters ................ 75
3.33 Clustering algorithm: first two line estimates. ..................... 76
3.34 Clustering algorithm: node assignment ........................... 77
3.35 Clustering algorithm: recomputing center line for North cluster ........ 77
3.36 Clustering algorithm: end of iteration 1 and line increment .......... 78
3.37 Clustering algorithm: start of iteration 2 ........................ 79
3.38 Manuscript HRC_leaf_M3, f96r: node clustering ............... 80
3.39 Ms. HRC leaf M3 f96r: bridge-segments ......................... 81
3.40 Three types of bridge segments .................................. 82
3.41 Erroneous line assignments: accent marks ....................... 83
3.42 Accents in the interlinear space: assignment heuristics ........... 84
3.43 Manuscript HRC_leaf_M3, f96r: assignment of internal and external segments 84
3.44 Manuscript HRC_leaf_M3, f96r: free segments and rings ........... 85
3.45 Manuscript HRC_leaf_M3, f96r: line separation .................. 86
3.46 Manuscript HRC_leaf_M3, f96r: line separation .................. 87
4.1 Segmented manuscripts .............................................. 93
4.2 Failed segmentation: Ms. MSB 23, with regular nodes .......................... 94
4.3 Failed segmentation: Ms. MSB 23, with pseudo-nodes ......................... 95
4.4 A typical mislabeling case: the "hanging mark" ................................. 96
4.5 A mixed component ........................................................................... 98
B.1 Ms. RARE FO.Z113.P3, University of Missouri, Columbia ................... 114
B.2 Ms. HRC 153, University of Texas, Austin ....................................... 115
B.3 Codex 1008, Dombibliothek, Cologne .............................................. 116
B.4 Codex 1068, Dombibliothek, Cologne .............................................. 117
B.5 Codex 1086, Dombibliothek, Cologne .............................................. 118
B.6 BANC MS UCB 85, UC Berkeley ...................................................... 119
B.7 MSB 23, John Hopkins University, Baltimore ................................... 120
C.1 Zhang’s algorithm: constraints for passes 1 (a-c) and 2 (a-b) ................... 122
C.2 $3 \times 3$ contexts for deletable pixels (pass 1 only - S and E erosion) ........ 123
C.3 $3 \times 3$ contexts for deletable pixels (pass 1 & 2 - corner erosion) ............ 123
C.4 $3 \times 3$ contexts for deletable pixels (pass 2 only - N and W erosion) ........ 124
C.5 Zhang and Zuen’s algorithm: failure cases ......................................... 125
D.1 The segmentation pipe-line: a simplified view .................................... 129
D.2 The segmentation pipe-line: complete view ....................................... 131
CHAPTER 1

Line segmentation techniques in off-line handwriting recognition

1.1 Statement of the problem

1.1.1 Historical handwritten documents: a challenge for the OCR field

According to current estimates, the number of surviving manuscripts of the 6th through the 15th centuries is 3.9 million. Most of them date to the late Middle Ages, a time when paper gradually replaced parchment, lowering the cost of book production. Only a minority of these documents are large-size illuminated volumes, with wide and well-formed characters and rules lines that have been pre-drawn with a lead point (Figure 1.1). Those Bibles, Books of Hours, literature or liturgy books, were usually paid for by rich patrons, stored with great care and rarely used.

By contrast, the bulk of handwritten documents is actually made of a relatively or-
ordinary material, whose content is no less valuable in the researcher’s eyes: collections of sermons, compilations and commentaries, many of them produced not in the monastery scriptoria, but in preachers’ convents and universities, the new centers of intellectual life after the 12th century. A great many of those manuscripts show

- tightly packed, often overlapping lines;
- skewed or fluctuating lines (Figure 1.3);
- a highly cursive style, in which characters, and even words, are chained together (Figure 1.2);
- small character sizes, that are no larger than a few millimeters;
- extensive use of shorthand, with numerous abbreviations in stenographic style (Figure 1.4).

Figure 1.2: Cursiva textualis script, Germany, XV (Joannes Andraee, *Super ar-borem consanguineitatis*)

Reading those pages is strenuous, slow work, even for well-trained researchers, which explains why most of these texts have not been edited to this day. Thus Optical Character Recognition (OCR) techniques have the potential to furnish a crucial tool to the reader, even if the possibility of fully automatic and robust systems seems remote.

Indeed, the very few studies on recognition of handwritten medieval Latin have an exploratory character. Edwards et al. [1] use a Hidden Markov model-based algorithm
to perform a full-text search in a 12th century manuscript of Terence’s Comedies. The test document was purposely chosen by the authors for its well-spaced lines and regular handwriting. Reddy and Crane [2] build on the open-source Gamera OCR infrastructure developed by Droettboom et al. [3]. In order to focus on the character recognition phase, the authors have to avoid any hard segmentation problems: separating the page into lines, words and characters is actually one the hardest tasks in optical recognition of historical handwritten documents.

1.1.2 Off-line vs. on-line recognition

Processing historical handwritten documents is an off-line handwriting recognition task: all data about the handwriting production steps are lost. By contrast, an on-line recognition process can store and use data associated with the generation of the written material: the direction and succession of the pen strokes, the ordering of the signs on the line and of the lines on the page. By comparison to an off-line process, the computational
and memory cost is relatively low, such that on-line OCR applications work reliably on handheld devices, for which they were designed in the first place. Among many others, a study by Ratzlaff [4] illustrates the on-line approach to segmentation: it uses temporal and spatial data to group the strokes into clusters, which are then themselves merged into lines. In the off-line approach, the recognition chain takes the completed writing as an input, which is stored as an image. Building a hierarchy among the graphical signs (characters, lines, block, columns) results from a posteriori inferences. When applied to historical handwritten documents, this task is especially challenging.

Nevertheless, the field of historical handwriting recognition is now a flourishing one. Notably, the countless projects involving non-Latin scripts (Indic, Arabic, Chinese) may be helpful to those who deal with older Western manuscripts. Although integrated projects drawing on existing techniques to build a complete OCR processing chain have their place in the field, most advances occur through narrower studies, which usually focus on only a few steps of the OCR pipeline. The present thesis is no different, focusing only on the problem of line separation in handwritten Latin manuscripts of the 15th century.

1.1.3 Line segmentation in off-line OCR treatments

The OCR pipeline A typical off-line processing pipeline is organized as follows (Figure 1.5 on page 5):

In the pre-processing phase, the input document, usually a color or a gray image, is treated as a whole and some filters are applied to it, depending on what is required by the algorithms of the next phase: for instance noise removal, background homogenization, and binarization (where color and gray pixel values are mapped on a black and white color scheme).

During the layout analysis, each document page is segmented into larger units, such as titles, columns, images, marginal annotations etc.

During the line segmentation process, blocks of handwritten text are separated into
Figure 1.5: The off-line OCR pipe-line
Segmentation into words and characters usually occurs at this point, but not necessarily in that order: a given algorithm might separate the characters before merging them into words. A number of segmentation techniques include a feature extraction step, during which the strokes are transformed into objects that are easier to describe. For instance, thinning reduces each letter to its skeleton.

In the character/word recognition step, each glyph -i.e. the specific shape that is under consideration- is assigned a character class. For example: "α" and "a" are both assigned the character class "a". This is typically done by pattern matching, where each glyph is compared to an existing template and given a score of similarity. This is often combined with other techniques, which make use of the linguistic context (word or sentence).

How to represent a text line There are several ways to code a handwritten text line when implementing the segmentation process. Likforman-Sulem, Zahour, and Taconet [5] list the following representations:

- the path that separates each text line from the adjacent line;
- the path that matches the baseline or the centerline of the written material (see section 1.2); this is usually an intermediary step, unless line identification has no other purpose than estimating a skew angle;
- the clusters of graphical components (connected components, glyphs, pixels) that are assigned the same line label;
- the lists of aligned units (connected components, blocks, glyphs)
1.2 Definitions

There is no unified terminology for handwriting description. The following definitions (see Figure 1.6) draw mainly on the terminology used by Petty [6], Likforman-Sulem et al. [5] and Ma and Doermann [7].

**Baseline** Imaginary line on which most characters (such as "a", "x", "o", "e", "r"...) rest.

**Median line** Imaginary line under which most characters (such as "a", "x", "o", "e", "r"...) are contained (see Likforman-Sulem et al. [5]).

**Centerline** Imaginary line that runs between the baseline and the median line.

**Core strip** The area delimited by the baseline and the median line (see Ma and Doermann [7]).

**Top strip** The area located above the median line; it is used for ascenders, accents and abbreviation marks (superscript letters, loops, etc.). The top strip of a given line often overlaps with the bottom strip of the adjacent line (see Ma and Doermann [7]).

**Bottom strip** The area located below the baseline; it is used for descendents. The bottom strip of a given line often overlaps with the top strip of the adjacent line.
Ascender  The part of a letter that extends above the median line, as in "d", "f", "l", "t".

Descender  The part of a letter that extends below the baseline, as in "g", "f", "p".

Overlapping components  Graphical units that belong to two adjacent lines and whose ascenders or descenders, respectively, share the same strip.

Touching components  Connected graphical units that encompass two or more consecutive lines, usually because ascenders and descenders join in the same strip.

Abbreviation marks and shorthand  Coming in every size and shape, they can be contained in the core strip, but usually occupy the top strip as superscript letters, accents, hats and loops. Some are separate graphical units, others are attached to the word they abbreviate.

1.3 Related work

The literature about handwritten text line segmentation has grown significantly since the last comprehensive survey of the handwriting recognition field by Plamondon and Srihari [8], most notably through projects about non-Latin scripts (Indian, as illustrated by Pal, Jayadevan, and Sharma [9], Chinese and Arabic). Once focused on short material (addresses, checks, signatures), the contributions often deal with extensive handwritten archives, where documents offer a wide variety of layouts and script variations. What follows is a selective review of the line segmentation literature dedicated to the handwritten historical documents. A few studies about print pages are included when deemed helpful in understanding the specific challenges and pitfalls associated with analysis of the handwritten line. Most studies tackle the specific challenges of the material at hand by combining a variety of techniques, which can be broadly categorized as follows.
1.3.1 Projection-based methods

These methods produce a 2-dimensional view of the document image by summing the pixels along the x-axis for each y coordinate. This technique has been often used to detect page layout (blocks and columns) and is more reliable for print documents than it is for handwritten material. When segmenting a binary image where ink strokes are coded with black pixels, local minima and maxima in the resulting histogram match the lines and the inter-linear gaps, respectively (Figure 1.7). A number of variations are commonly used:

- the horizontal axis pixel sum can be replaced by a pixel average, by a white-pixel-to-black-pixel transition count (or run number), or by the number of connected components in the line, both of which take into account the number of times a given pixel line intersects with a stroke, thus providing useful information about the distribution of the ink along a given line (see Kołcz et al. [10] or Bruzzone and Coffetti [11]);

- after delimiting vertical strips in the page, partial profiles are generated for each strip; it is a straightforward approach, albeit not very robust, to the segmentation of skewed lines, illustrated by Weliwitage et al. [12] and Cohen et al. [13], among others.
The project by Bruzzone and Coffetti [11] applies the profile technique to the line segmentation problem. The authors identify peaks and valleys in vertical projections to delimit respectively the line bodies and the white strips separating the lines. Then they assign the connected components to the lines. As with many earlier studies in handwriting recognition, the work is focused on small handwritten blocks (addresses).

Weliwitage, Harvey, and Jennings [12] first estimate baseline positions from vertical projections. Then they compute a cost function for neighboring rows that takes into account the distance from the estimated line start row, the cutting of foreground (ink) pixels for the row and the deviation of an hypothetical segmentation line joining the row under consideration with the start row. The best line intersects with the row that minimizes the cost function.

Papandreou and Gatos [14] innovate by using horizontal projection profiles (i.e. a histogram of column pixels values along the x-axis) to detect skewed blocks of text. They assume that the vertical strokes, present in a majority of characters, are the chief contributors to the distribution peaks. The rotation of the text lines that maximizes the projection peaks gives the angle of a proper alignment.

In a study by Kavallieratou, Fakotakis, and Kokkinakis [15] skewed text lines are aligned by maximizing the peaks of the vertical projection. The authors do not rely on the profile itself to spot the correct angle, but on a transform of the projection (the time-frequency Wigner-Ville Distribution).

Kołcz, Alspector, and Augusteijn [10] estimate the line location by computing a Fourier spectrum of the vertical projection. Looking for the frequency that contributes the most to the signal gives the period, i.e. the line spacing. Horizontal projections in each line strip are then used for word-spotting.

In profile-based methods, the choice of parameters (smoothing bandwidth, thresholds) can be critical when dealing with handwritten lines: the profiles are much noisier
than projections created from print pages. Using a transform (a choice made by Kavalieratou et al. [15], as well as Kołcz et al. [10]) might be the most robust way to extract informations from a histogram.

1.3.2 Smearing methods

In this family we place all methods that make the document image appear as if it were seen from a distance, where small gaps –like white spaces between characters and words– disappear and larger gaps –like interlinear gaps– remain, thus providing a simplified view of the document (Figure 1.8), that can be used as a filter to isolate the lines: separate graphical units are then assigned to line areas, by an ‘AND’ operation. There are many ways to realize this, as suggested by the two following examples.

Li et al. [16] obtain an initial estimate of the lines by blurring the text with an anisotropic kernel (thus preserving gaps between lines while reducing white spaces inside each line). In the next phase, the image is considered as a 3-D curve, with ridges whose
closed boundaries are to be determined. A level-set algorithm controlled by a partial differential equation iteratively computes optimal boundaries for the line blocks. Heuristics about block collisions help with convergence. The method requires a substantial post-processing phase, but it can deal with intricate freestyle documents, such as casual notes.

The algorithm described by Nikolaou, Makridis, Gatos, Stamatopoulos, and Papa-markos [17] creates homogeneous regions in a binary image by replacing a sequence of background pixels with foreground pixels if the number of background pixels in the sequence is smaller or equal than a predefined threshold. Their Adaptive Run Length Smoothing Algorithm improves on a previous one when dealing with non-homogeneous letters and differently slanted lines. To make better decisions about erasing the background pixels, the connected components nearby are checked for overlapping and size, so that overlapping components are not merged into a single line unit. Optimized for historical print material (Greek), the method is robust enough to deal with skewed text and character size variation.

1.3.3 Grouping methods

This category embraces methods that construct lines through a bottom-up process, starting with sub-line units (salient features, characters, connected components, blocks) that are then aggregated along hypothetical line axes.

Liolios et al. [18] apply the clustering principle to the estimation of skew angles in handwritten documents. Connected components are regrouped in clusters by looking for the nearest neighbors along an estimated line axis; then the line axes are corrected by a least-square fit along the clusters. Since the authors only look for an estimate of the overall skew angle of the page, they simply prune the underpopulated clusters (e.g. incomplete lines) in order to select the line estimates whose angle is likely to be closer to the page angle. When line identification is a goal in itself, cluster pruning is not an option. Moreover, the nearest-neighbor approach to clustering works when lines are well separated. The assumption that connected components in the same line are closer to each
other than they are to connected components located in an adjacent lines does not apply to handwritten historical manuscripts. A robust clustering technique for handwritten medieval manuscripts should bypass the connected component analysis, in order to consider more reliable features of the script line.

In their research, Gupta and Chanda [19] first label the connected components and build their bounding boxes. If they are found to have enough vertical overlapping, successive boxes are then linked together by their centroids. Line construction starts from the uppermost box in the page, examining boxes in the order of their vertical coordinates and linking together the boxes that have sufficient vertical overlap in a tree structure. The resulting paths match the centerlines of the text lines (Figure 1.9).

Albeit elegant, the approach relying on Voronoi diagrams to identify line clusters is not robust enough to deal with handwritten texts. It allows for a fast segmentation of complex print pages in a study by Kise et al. [20]: a Voronoi point diagram is constructed from sample points in the characters, then simplified by erasing the boundaries inside each
connected component. Line segmentation is performed by selecting the correct edges in the resulting Voronoi area diagram, with the help of a neighbor graph. This approach works well on print text lines, with well separated characters, but is likely to fail on cursive handwritten lines, where the connected components vary wildly in shape and size, often belonging to two or more lines.

1.3.4 Hough Transform

The Hough transform (see Hough [21]) is one of the most popular techniques used to detect alignments in a document.

Each line that goes through a point \((x, y)\) of the plane can be represented by a point in the plane \((r, \theta)\), with \(r\) and \(\theta\) the polar coordinates of the line. The set of lines going through \((x, y)\) is a sinusoid in the \((r, \theta)\) domain. Reciprocally, the set of sinusoids that intersect at \((r, \theta)\) matches a set of colinear points in the \((x, y)\) space. Once an image has been transformed in the Hough space, it can be tested for local maxima created by numerous sinusoids intersecting in the same location: those points provide the parameters for existing alignments in the original document.

In our perspective, a typical handwriting line detection algorithm would first consider a set of points potentially belonging to the same text line and then scan the Hough domain for alignments that would fit those points (an approach illustrated by Fletcher and Kasturi [22], for instance). Most enhancements to this approach aim at reducing the cost of this computationally intensive procedure.

1.3.5 Mixed methods, other methods

Many, if not most, studies do actually combine a variety of techniques on the material of choice.

Nikolaou et al. [17] use an estimate of the letter height to blur out white spaces in the lines, then trace two series of horizontal, left-right paths: paths that separate the lines by
following the optimal (lowest) route between the N and S ridges; centerline paths on the ridges themselves. Connected components, labeled independently in a previous treatment, are then assigned to line areas.

Zahour et al. [23] try to avoid the pitfalls associated with text line metrics (height, spacing) derived from the bounding boxes of the connected components, especially when dealing with scripts that make heavy use of the line top strip (accents, diacritic marks). They partition the Arabic text into vertical strips of equal width, and then delimit writing blocks in them, by the vertical projection method (Figure 1.10) The goal is to isolate a high percentage of non-overlapping, non-touching writing fragments, so that line metrics can be derived in a more robust way. Multi-touching blocks can then be identified and split in a reliable way and lines labelling can be performed. It is not a parameter-free procedure: a training phase allows classification of texts among a variety of block patterns, so that an optimal block width is determined for each text.

1.3.6 Remarks

A few considerations on the studies reviewed above will help frame the current work.
1.3.6.1 Dependence on user-defined parameters

Many procedures rely on initial parameters that have been found to do well on the test material: thresholds for local maxima detection in profile-based methods (Bruzzone and Coffetti [11]), enclosing box thresholds (Gupta and Chanda [19]), or document-specific metrics: Nikolaou et al. [17] use some assumptions about the script (printed Greek) to determine a constant factor for possible variations in character size in otherwise homogeneous units (for example, the heights of a "ρ" and a "γ" in the same line should not differ by a factor larger than 3.5). Similar heuristics and assumptions might not be possible nor recommended when processing highly cursive and noisy handwritten lines, especially late medieval scripts.

Most studies have to find a balance between genericity (i.e. the ability to yield satisfactory results on a wide range of handwritten documents) and performance. Even when word and/or character segmentation is not in the scope of the study, a script-specific approach is often the best way to minimize the error rate. It helps explain why the automatic recognition of Indic scripts alone is a extensive field in itself, as shown by the recent surveys conducted by Hole and Ragha [24] and Pal et al. [9].

By focusing on a relatively narrow family of scripts, the technique described in this thesis takes advantage of script-specific features and minimizes the reliance on user-entered parameters.

1.3.6.2 Extracting metrics

Statistics computed from the material at hand help minimize the reliance on pre-set parameters.

Most text line segmentation methods need at least some rough metrics for the line height: it can be an average character height estimate (Louloudis et al. [25]), or a line spacing estimate: Kołcz et al. [10], for instance, use the Fourier spectrum to derive the line period; Gupta and Chanda [19] rely on the average height of the connected components
enclosing boxes to detect larger, 2-line encompassing boxes. Nikolaou et al. [17] compute the average character height from statistics about the bounding boxes to make informed decisions about grouping graphical units.

Our segmentation technique relies on a few metrics, obtained by methods that yield stable results over a wide range of manuscripts.

1.3.6.3 The role of the connected component analysis

The connected component analysis is a classic step in many line segmentation procedures, used as a way to identify non-touching, identically valued regions in the page: typically, distinct sets of foreground pixels. In printed documents, most connected components are single letters; in handwritten pages, the components more often match an entire word, and even groups of characters that cross the word boundary. Simple metrics can be extracted from the components’ bounding boxes, to help estimate the line height. Components are also easily manipulated, and often provide a convenient way to work on an intermediate scale, right between the text block scale and the character level. They are a very manageable population of graphical units, often chosen as a foundation for bottom-up segmentation.

However, relying on connected components can lead to problems in late medieval script processing. The cursive style and the tight packing of the lines give way to numerous touching (line-crossing) components. On narrow columns or shorter lines, the component population is too small to yield reliable metrics about the line axis;\(^1\) hence the need to find graphical features that describe the handwritten line better without introducing too much noise.

This thesis explores a segmentation method in which connected component analysis plays only an auxiliary role in the chain. Line detection in particular does not rely on it.

\(^1\) Louloudis, Gatos, Pratikakis, and Halatsis [25] are clearly aware of that pitfall when they partition each connected component into several sub-components, in order to create more voting members for a Hough transform analysis.
**Figure 1.11:** Cut pixel minimization path [12]: the path avoids ascenders or descenders by attempting to go around them unless the row penalty is too great.

directly.

### 1.3.6.4 Dealing with touching characters

Characters that touch the adjacent line may be the most common cause of failure in line segmenting algorithms. The connected components analysis, so efficient at isolating the characters in the print page, is a crude tool when applied to historical handwritten material, as demonstrated by the range of strategies that address this problem.

Bruzzone and Coffetti [11] suggest that contact points in touching characters could be detected by analyzing the geometrical features of the strokes in the area of interest. A corner-detection algorithm would identify potential cutting locations. Tellingly, this latter phase of the project is not implemented and the separation is simply performed along the line strip boundaries retrieved in the previous phase. The algorithm devised by Weliwitage et al. [12] tracks around the ascenders and descenders (Figure 1.11) using a cost function to penalize the deviation from the inter-linear axis. If the cost is too high, as it is when dealing with touching components, the algorithm cuts through the letters. Nicolaou and Gatos [26] adopt a similar solution, cutting the connected components where separating paths cross them. In the study by Gupta and Chanda [19], connected components that exceed a given threshold are split into as many strips as necessary, then assigned to their respective lines. In these three algorithms, the cut occurs where the letter intersects with the segmenting line (see 1.3.1), offering no guarantee that the letters’ integrity is preserved in either of the resulting lines.

Guessing reasonable locations for the cut is not a trivial task. In the more elaborate
strategies, it requires dedicated character segmenting techniques. For example, Louloudis et al. [25] detect 2-line high connected components, then explore the character skeletons for junction points located in the intra-linear space. Kang and Doermann [27] propose a much more sophisticated, template-based approach to separating touching components. Local touching strokes are isolated in patches (Figure 1.12) that are then compared to a dictionary associating local touching patch (LTP) templates and their correct segmentation. The dictionary was built in a previous training phase.

Our study devises a way to separate the touching components, by clustering their constituting parts around the centerlines axes.

1.4 Scope of the current work

1.4.1 Sample documents: "hard" manuscripts from the 15th century

This work is an attempt at segmenting lines in "hard" handwritten documents from the late Middle Ages.

The documents used as a sample input for this investigation (see Appendix A.1) are separate folios from a compilation of Simon of Cremona’s² sermons ([28], [29]). The scribe is Caspar Minensis, from Buxheim, Germany, who collected these sermons between 1425 and 1450 (the exact date is unknown). The digital files were retrieved from Columbia University’s Digital Scriptorium [30], as 2000 × 2520 pixel color images, in JPEG compressed format. The evaluation phase of this thesis employs an enlarged set of manuscripts described in detail in Chapter 4. All documents meet the following criteria:

²A celebrated preacher who died in Padua in 1390.
• the script size is small, with the average character height not larger than 3 mm; the
resulting thickness of the pen strokes -with respect to the glyph size- often obscures
the pattern;

• the lines, about 5 mm high, overlap and contain numerous touching characters;

• the line spacing varies;

• some lines are skewed (±2.5°); a few of them are fluctuating;

• the writing style is loose, using a cursive Gothic bastard script;

• the top strip of each line contains shorthand symbols and other abbreviation marks.

Each of these features amounts to an obstacle to correct line segmentation. This work
explores some solutions to these problems and describes a clustering-based algorithm for
line separation.

1.4.2 Constraints

We define the following constraints for this study:

1. line segmentation in a column of text is to be based on any line representation deemed
relevant to the task: centerline paths, separating paths (see 1.1.3), provided it can be
used to obtain a separate image of each detected line;

2. the algorithm should be flexible enough to deal with lightly skewed lines (± 5°) and
incomplete lines;

3. the algorithm will be evaluated by metrics about: the detected text lines, the cor-
rect assignment of the components to the lines, the quality of the cut into touching
components;
4. the line segmentation process should not be too dependent on the image dimensions; given a specific document, it should perform equally well on a 2000x2500 pixel representation and a 4000x5000 pixel representation.

5. although the work is focused on late medieval script and uses Latin manuscripts as test material, it should be relevant to all cursive scripts of the period (15th century), including documents in vernacular languages: it is not language-specific.

6. documents have been chosen so that simple noise removal and binarization techniques produce adequate input for the segmentation process; in some degraded manuscripts, more sophisticated pre-treatments might be required to take care of partially faded characters.

7. even if the computation of the text block boundaries is performed here as a straightforward application of the profile method, the analysis of complex layouts (involving marginal annotations or a general skew in the page, for instance) is not a goal for this work;

8. while some work is done at character-level, especially when dealing with touching components, character description is not a goal in itself.

Some design choices result from these goals:

• no training phase, nor manual setup should be necessary for correct execution of the algorithm;

• input parameters (e.g. thresholds, strong assumptions about late medieval Gothic script geometric features) are avoided.

This study begins with an application of the profile-based approach to the test material (Chapter 2). Even if it comes short of meeting the requirements exposed above, it
helps with choosing the right method for text binarization, a treatment necessary for all subsequent transformations. Fourier analysis of the profile provides an estimate of the line spacing, an important statistic for the segmentation procedure. Chapter 3 is dedicated to the two phases of the line segmentation process: section 3.2 exposes how the document image is transformed in order to extract salient morphological features, such as pen strokes; section 3.3 details the method used for line separation, which follows a bottom-up, clustering-based strategy. Chapter 4 describes the testing protocol and a general evaluation of the segmentation procedure.
CHAPTER 2

Line segmentation with the profile method

The treatments exposed in this section belong to the preprocessing phase. Since the color document image is not a suitable input for any segmentation program, experiments with the most commonly used thresholding methods enable us to devise an automated binarization chain (Section 2.1). A few metrics about the handwritten lines are then derived from the vertical projection of the binarized image: although the automated detection of line locations proves difficult (Section 2.2), determining the text block boundaries is both feasible and useful for our purpose (Section 2.2.3); an analysis of the profile’s frequency domain, the Fourier transform, is a reliable means to compute the average interlinear spacing, a data crucial to our line segmentation method (Section 2.3).

2.1 Binarization problems

2.1.1 Properties of the document images

2.1.1.1 Color coding

The document image used for this experimentation is fairly representative of the material usually stored in digitized manuscript libraries. Because color is now seen as a crucial component of any (even non-illuminated) manuscript, it is de rigueur in the digitized document. The format matches the constraints of a database designed for online use: lossy JPEG compression, three-channel color coding (sRGB), \(^3\) 300 dpi. Although any document segmentation task can be made easier by the choice of a format that better preserves

\(^3\)Document images created for printing usually use the CMYK colorspace.
The color histograms (Figure 2.1) give the pixel count for the red (R), green (G) and blue (B) color channels. Zero valued components of pure black pixels have been filtered out. The particular quality of lighting in the bright, peripheral parts of the page explains the high pixel count with a red component at its maximum value (255). Not taking into account the extrema, the distribution is bimodal, a typical feature of document images, which are defined by a strong differentiation between foreground and background components.

**2.1.1.2 The manuscript’s palette**

Foreground pixels representing the inked portions of the page are mostly dark valued pixels. The ink is a dark brown pigment, with varying opacity, depending on the location in the pen stroke: pure black pixels are concentrated on the character edges, while the middle part of the strokes and many accents are slightly transparent. This can be made more obvious through a good color quantization method, such as the median cut algorithm (Heckbert [31]), which recursively slices the color cube in its longest component, each time at the point representing half of the pixel population for the color range under consideration. Even with a small number of colors, this approach reveals small brightness variations in regions that otherwise would look uniformly black, as shown by Figure 2.2.
The manuscript page contains numerous annotations in red. Even in fairly ordinary documents, like this one, it was customary for medieval scribes to highlight some initial letters, most biblical quotations, punctuation marks and the various names for God with red marks (see \textit{\textit{xps}} – for \textit{\textit{\chi\rho\sigma}} (\textit{\textit{Christus}}), for instance).

The background is highly textured, a common feature of many lesser quality medieval papers. Lines written on the back are visible through the paper, contributing to the greyish tones of the interlinear regions. Otherwise there are no stains or faded portions.

2.1.1.3 Paths to binarization

The binarization step must preserve as much as possible of the near-black foreground patterns while minimizing noise. Since red annotations are not deemed essential to the document interpretation and are a potential obstacle to character analysis, their removal is desirable. Even if our test document, with its well-differentiated background and foreground, is not a particularly challenging case, the choice of a binarization technique is not obvious. The paths explored by this study are summarized in Figure 2.1.1.3.
2.1.2 Working with a grayscale document

Using a grayscale version of the document image as an input for the binarization process makes poor use of the information contained in each color channel: averaging the R, G, B components into a single channel deletes data that could otherwise help fine-tune the binarization step. Grayscale coding deserves nonetheless a short detour in this exposition: indeed, working with a grayscale document in a preliminary stage allows us to better compare the two existing approaches in text binarization: the global and adaptive thresholding methods.

2.1.2.1 Global thresholding (Otsu)

Document binarization transforms 8-bit pixels (256 colors) into 2-bit, i.e black (0) or white (1) pixels. Each pixel value is compared to a threshold value, then shifted accordingly to 0 or 1. Assuming that the manual choice of a threshold value is not practical for automatic document analysis, Otsu’s method (Otsu [32]) allows for automatic setting of a unique value, used thereafter for each pixel in the image.
In this method, the threshold is optimal when the two groups of pixels in the resulting image match two groups in the input image that are as homogeneous as possible. Therefore an optimal threshold value minimizes the sum of group variances, or within-group variance

$$\sigma^2_W(t) = q_1(t)\sigma^2_1(t) + q_2(t)\sigma^2_2(t)$$

where $q_1(t)$ is the probability of those pixels whose values are less than or equal to $t$ and $q_2(t)$ the probability for those pixels with values greater than $t$, and $\sigma^2_1(t)$ and $\sigma^2_2(t)$ the respective variances of groups 1 and 2.

The total variance is the sum of the within-group variance and the between-group variance $\sigma^2_B(t)$:

$$\sigma^2 = \sigma^2_W(t) + \sigma^2_B(t)$$

Since $\sigma^2$ is a constant and does not depend on $t$, the value of $t$ that minimizes $\sigma^2_W(t)$ is also the value that maximizes $\sigma^2_B(t)$. Rewriting $\sigma^2_B(t)$, we obtain

$$\sigma^2_B(t) = q_1(t)q_2(t)[\mu_1(t) - \mu_2(t)]^2 = q_1(t)[1 - q_1(t)][\mu_1(t) - \mu_2(t)]^2$$

where $\mu_1(t)$ and $\mu_2(t)$ are respectively the means of the first and second groups.

From the relationship $q_1(t + 1) = q_1(t) + P(t + 1)$ (where $P(t)$ is the probability of pixels of value $t$, $\mu_1$, $\mu_2$, and consequently $\sigma^2_B(t)$ are obtained by recursion for each $t$. An iterative implementation of this method computes the optimal threshold $t$ at a small cost.

Otsu’s method works well with images that match (even approximately) the initial assumptions of a bimodal distribution of the pixel values, which is the case for most written document images. Figure 2.4 shows a grayscale version of the manuscript binarized with this method. The method discriminates well between inked portions of the manuscript and background components, with no loss of information and almost no noise. As expected

---

4See Shapiro and Stockman [33] for a detailed exposition.
Figure 2.4: Otsu automatic thresholding operator

with a grayscale input, the red annotations of the original image are not discarded in the process.

2.1.2.2 Adaptive binarization (Sauvola)

Adaptive thresholding techniques address some limitations of the global threshold method. With a threshold value that varies across the image, local background variations are taken into account. The threshold is usually a function of the mean pixel value for the neighborhood. A straightforward application of this principle is exposed by Niblack [34], who computes a variable threshold for a pixel \( P \) as:

\[
t(P) = k\sigma(N_n(P)) + \mu(N_n(P))
\]

where \( \sigma(N_n(P)) \) and \( \mu(N_n(P)) \) are respectively the standard deviation and mean of \( P \)'s \( n \) neighbors and \( k \) is a constant, user-entered parameter. While Niblack’s focus was on astronomy photography or X-ray images, Sauvola and Pietikäinen [35] adapt the method to document images with the following formula:

\[
T(row, col) = \mu(N(row, col)) \times [1 + k\left(\frac{\sigma(N(row, col))}{R} - 1\right)]
\]
where the contribution of the standard deviation to the local threshold value is multiplied by the mean, thus varying from one location to another. The standard deviation term is also weighted by R, a quasi-constant, whose value is typically 128 (or half the range of the pixel values). The computational cost of Sauvola’s method varies greatly with the size of the neighborhood window used for each pixel. Our implementation computes the threshold for sample pixels\(^5\) through the image and interpolates the remaining values. Figures 2.5(a)–(f) show the output of Sauvola’s adaptive thresholding method with different sets of parameters.

User-defined parameter \(k\) has a significant impact on the quality of the output. The possible benefits of an adaptive method come here at the expense of automation, which does not fit our goal of a parameterless document processing chain.

Moreover, Sauvola’s method is only as good as the sliding window is large. This is particularly obvious with scripts made of relatively thick strokes, where the adaptive op-

\(^5\) Every five pixels.
Figure 2.6: Ms. HRC leaf M3 f96r: horizontal black run lengths

The adaptive thresholding operator discriminates between tones that should be aggregated instead, resulting in cavities and edge contouring. This is particularly visible in Figure 2.5(d). The hypothesis that the optimal size for the window depends on the stroke width is a reasonable one. Nevertheless, it is not the goal of this thesis to test it thoroughly. Median stroke width can be approximated by counting black run lengths. Confronting the output of Figure 2.5 with the stroke width statistics (Figure 2.6) suggests that the threshold computation window has to exceed the average stroke width by a factor of two or three to give acceptable results.

In short, the drawbacks of Sauvola’s adaptive thresholding make it impractical for our binarization needs. On a well-preserved manuscript, a global approach yields better results at a small computational cost, with the benefits of a fully automated process. The next section describes the adaptation of Otsu’s operator to the binarization of a color image.

---

6 The print documents binarized by Badekas et al. in their comparative study [36] show similar problems, suggesting that Sauvola’s method is not a good fit for documents containing large black patterns.

7 The image is scanned horizontally and/or vertically. A black run is a continuous sequence of black pixels within the line (column).
2.1.3 Working with the color image

By a distinct treatment of each color channel, a selection is made among the inked portions of the manuscript. The method consists of two stages:

1. each channel is thresholded separately, effectively reducing the 24-bit colorspace to a 3-bit colorspace (8 colors); the threshold value is determined separately for each channel, using Otsu’s operator.

2. All unwanted colors are then merged with the white background.

2.1.3.1 Color reduction

Where a sensible color quantization algorithm makes use of the actual colors in the image to reveal otherwise invisible variations in ink opacity (see Section 2.1.1.2), the color reduction procedure considered at this stage is a straightforward operator that sets each pixel’s channel to its maximum value if it is equal or greater to the threshold value $t$, and to 0 otherwise. Also called posterization, this procedure transforms a pixel composed of three 8-bit coded colors into a pixel composed of three 1-bit colors. Shifting each pixel’s component by 7 positions to the right\(^8\) is the most straightforward way to posterize an image. Instead, Otsu’s method is used here to compute first a threshold value for each of the R, G, B channels. Then, each color is thresholded separately, giving the images shown in Figure 2.7.

This procedure has been found to work well even on difficult manuscripts, like the document featured in Figure 2.8(a). This manuscript (Ms. RARE FO_Z113_P3 [29]) has been produced by the same hand as Ms. HRC leaf M3 and was originally part of the same volume, but is now kept in another library. The diluted ink gives a transparent, watercolor-like quality to the characters. A shown by Figure 2.8(b), our color reduction process discriminates correctly between the transparent-brown patterns and the red annotations, while

---

\(^8\)Which amounts to a threshold equal to half of the color range, i.e. 128.
Figure 2.7: Reduction to 3-bit colorspace (8 colors): (a) Ms. HRC leaf M3, f86r (detail) (b) Channel-specific color thresholds: note that the red annotations are clearly separated from the black characters.

Figure 2.8: Reduction to 3-bit colorspace (8 colors): (a) Ms. FO_Z113_P3 (detail) (b) With binarized color channels minimizing the loss of information.

A simpler method computes a single threshold with Otsu’s operator from a gray-tone image, to be applied to each channel in the following step. The output varies according to the gray-tone conversion method chosen. In Figure 2.9(a), the grayscale image used to compute the threshold is obtained by averaging the most prominent and less prominent channels of each pixel. In Figure 2.9(b), the grayscale image used for the computation is the average $\mu = .21R + .72G + .007B$, also known as the luminosity-based grayscale con-
Figure 2.9: Reduction to 3-bit colorspace (8 colors), from a single threshold: (a) The threshold is computed with a gray-tone image which is the average of the most prominent and less prominent channels of each pixel (b) The threshold is computed with a gray-tone image which is a weighted average of the 3 channels.

version. The 3-bit resulting output is significantly worse in the latter case, with a pervasive red component eating at the foreground patterns.

Channel-specific thresholds seem to produce more consistent results. Therefore it is the method of choice for the subsequent phases of this work.

2.1.3.2 Production of a binary image

Obtaining a monochrome image from the 8-color file is a trivial task, that can be completed by merging every color except white (111) and black (000) with the white background component. The binarized manuscript image is shown in Figure 2.10.

There is no reason at this point to refine the binarization process further. The color reduction approach gives very usable input for the segmentation procedure exposed in this thesis. The quick experimentation conducted in the present section suggests that an adaptive technique is only marginally helpful to binarization of well-preserved manuscripts.
2.2 Profile smoothing

2.2.1 Local linear kernel smoothing

The vertical pixel value projection or vertical profile of the document image is obtained by summing up foreground pixels for each row. The resulting plot is shown in Figure 2.11. Although its roughly periodic character is immediately apparent to the observer, with the main peaks and valleys matching the core strips and interlinear spaces in the pages, the signs and accents that populate the upper strip of each line create some irregularities. In order to use the projection as a base for line detection, all secondary peaks have to be smoothed out.

One way to smooth the profile is to convolute it with a local distribution function or kernel. The red, noise-free curve in Figure 2.11b has been obtained by local linear regression: at each point of the profile, a straight line $\hat{\beta}_0 + \hat{\beta}_1(x)$ is fit to the $y_i$ using weighted least squares, where the weights are chosen according to the height of a normal distribution centered about each of the $x_i$ (the kernel). The local linear kernel estimator $\hat{m}(x; h)$ is found by minimizing the error function

$$\sum_{i=1}^{n} \{y_i - \beta_0 - \beta_1(x_i - x)\}K_h(x_i - x)$$
where \( K_h(x_i - x) \) is the weight function, a kernel scaled by a bandwidth \( h \). An explicit formula for the estimator can be found in Wand and Jones [37].

As seen on Figure 2.12, the bandwidth \( h \), i.e. the scale of the kernel weight function, is critical to the result. Too small a window will result in a wiggly estimate, where the interlinear noise is still present (Figure 2.12(a)); too large a window smooths out the line-

\[ \hat{m}(x; 1, h) = n^{-1} \sum_{i=1}^{n} \frac{(\hat{s}_2(x; h) - \hat{s}_1(x; h)(x_i - x))K_h(x_i - x)Y_i}{\hat{s}_2(x; h)\hat{s}_0(x; h) - \hat{s}_1(x; h)^2} \]

where

\[ \hat{s}_r(x; h) = n^{-1} \sum_{i=1}^{n} (x_i - x)^r K_h(x_i - x) \]
dependent variations which we intend to detect (Figure 2.12(b)). For our sample document, any value between 5 and 10 pixels is an appropriate choice, that selects only those maxima that match a written line (Figure 2.12(b)).

Then, accurate line metrics can be obtained easily, in 3 steps:

1. compute the median $m$ of all profile ordinates $y_i$

2. use $m$ as a threshold value for the profile: each ordinate $y_i > m$ is assigned an arbitrary positive value, while the remaining ordinates are set to zero (Figure 2.13)

3. the number of lines is the number of positive run lengths through the thresholded profile; an estimate of the line-to-line spacing is the median of the run lengths.

**Figure 2.12:** Smoothing a profile by local linear kernel regression: (a) Secondary local maxima remain in the smoothed profile (b) The profile is oversmoothed.
2.2.2 Kernel smoothing is not easily automated

Although this procedure can yield reliable line metrics (in this case, a 40 pixel inter-linear spacing), its dependency on the user-defined parameter $h$ is a significant drawback. As a matter of fact, the optimal kernel size depends on the average line spacing, the very feature we intend to detect through the smoothing procedure. The same bandwidth that has been found to work well on a given document image is not likely to give satisfactory results on a script characterized by different metrics or even on a file created with different scanning settings. For instance, Figure 2.14 shows the application of a kernel of size 10 to the manuscript FO_Z113_P3 (see section 2.1.3.1). Although it was a part of the same original codex, the character and line metrics are slightly different: the hand is looser, resulting in significantly skewed lines and increased line spacing (about 52 pixels vs. 42 pixels). The resulting profile is oversmoothed. Even if a correct line spacing estimate can still be obtained, the line count is incorrect.

The choice of an optimal value for $h$ cannot be easily automated. Some bandwidth selectors that have a correct asymptotical performance do not work well with limited amount of data. Every selection method has to find a compromise between bias (how far away the computed bandwidth is from the optimal bandwidth) and variance (the spread of

---

10See Wand and Jones [37], pp. 59-60 for an introduction to the problem of bandwidth selection. As shown by Köhler et al. [38] in a more in-depth exposition, bandwidth selection methods are rapidly becoming a field in itself, with highly technical problems, beyond the scope of the present thesis.
Figure 2.14: Profile-based line detection in Ms. FO.Z113.P3: (a) The vertical projection sums up the black pixels in each row (b) Projection smoothed with by local linear regression (bandwidth=10): some lines are merged.

The vertical projection will be used only for those computations that do not rely on user-entered parameters. The profile thresholding method is a straightforward way to determine the top and bottom boundaries of the text block to be segmented. The vertical projection is first binarized, using the median value as a threshold. The top boundary is located at the transition between the initial zero-valued region (which matches the upper

2.2.3 Profile-based estimate of the text block boundaries

The vertical projection will be used only for those computations that do not rely on user-entered parameters. The profile thresholding method is a straightforward way to determine the top and bottom boundaries of the text block to be segmented. The vertical projection is first binarized, using the median value as a threshold. The top boundary is located at the transition between the initial zero-valued region (which matches the upper
margin) and the first local maximum. The bottom boundary is located at the transition between the last non-zero row and the zero-valued region corresponding to the lower margin. Using just the raw projection as an input, this procedure yields satisfactory estimates for the text block boundaries, with no need for user-set parameters. Figure 2.15 shows the result of this procedure on various manuscripts, with the boundaries slightly corrected, to better fit the needs of our segmentation algorithm (see Section 3.3.1). As illustrated by Figure 2.15(a), this method is robust enough to deal with outsize glyphs that extend well into the margins, such as initials (Figure 2.15(b)) or ornate letters (Figure 2.15(b)).

Moreover, a metric crucial for the rest of this work can still be obtained by analyzing the vertical projection, albeit with a different approach. While the smoothing procedure deals with a function of the image rows, Fourier analysis deals with the image’s frequency domain, allowing us to retrieve the main component of its nearly-periodic line patterns in a very reliable way.

2.3 Estimating the interlinear spacing by Fourier analysis

The vertical projection profile of the handwritten lines behaves like a periodic signal, albeit a very noisy one. Although an adequate estimate of the distance between two peaks can be obtained by smoothing out the noise (see Section 2.2), the choice of an appropriate smoothing parameter is not easily made by an automatic procedure. By contrast, Fourier analysis provides a reliable way to isolate the fundamental frequency of the signal, and thus to compute its period.

Let us consider a \( N \)-row window in the vertical projection profile. For each \( y \)-coordinate in the image, the projection value is a sample of a function \( f(\rho) \) we would like to estimate, with \( \rho \) the sampling interval, i.e. 1 pixel. By definition (see Bracewell
Figure 2.15: Block boundary computation: the green vertical lines mark the column boundaries. The slight offset with respect to the thresholded profile is due to a correction made necessary by the clustering algorithm of Chapter 3.
Figure 2.16: Ms. HRC leaf M3, f86r: Fourier power spectrum of the vertical ink profile.

[39]) $f(\rho)$ possesses a discrete Fourier transform $F(\nu)$

$$F(\nu) = N^{-1} \sum_{\rho=0}^{N-1} f(\rho) e^{-i2\pi(\nu/N)\rho} \quad \text{for} \quad \nu = 0 \text{ to } N - 1$$ \hspace{1cm} (2.1)

where $\nu$ is the frequency integer, with $\nu = Nf$.

We thus represent $f(\rho)$ in the frequency domain by a Fourier series consisting of a constant term (the mean of the values considered) and multiples of a fundamental frequency. By the Nyquist sampling theorem, the highest frequency needed to represent the signal will be the $\frac{N}{2}$th harmonic term, that is, 0.5 hertz, which corresponds to 2 samples per period.$^{11}$ Such a representation could theoretically detect those components of the projection whose period is only 2 pixels, it is more than adequate to recover a low frequency component, such as the regular succession of handwritten lines. Since the top and bottom margins of the page are not part of the vertical projection profile under consideration, the transform is computed from the entire data set. Even if a smaller record length (a 512-

$^{11}$The sampling rate must be at least twice the frequency of the highest frequency component to be recovered in the signal being sampled (see Bracewell [39]).
pixel window, for example) would provide a very similar spectrum, using all data makes the resulting estimate less sensitive to local variations of the line spacing over the written page. The energy spectrum $|F(\nu)|^2$ of the transform (see Bracewell [39]) is a convenient way to synthesize Fourier complex coefficients when working on physical values\(^{12}\) (Figure 2.16). The frequency corresponding to the peak in power (i.e. to the dominant cyclic variation in the page ink patterns along the y-axis) is 0.025, which gives, for example, a period $p = 40$ pixels ($1/0.025$) for Ms. HRC leaf 53, f96r.

The line period computed with this method is a reliable statistic that can be used to estimate the position of a line with respect to a line detected in a previous step. Because it masks the interlinear spacing variations that can occur across the page, the line period must be used with caution. The line detection algorithm described in the section to follow will use it to initialize the segmentation algorithm, and then replace it with better estimates computed from the already detected lines.

\(^{12}\)Some information is lost in the process, that is, the phase angle of $F(\nu)$. 
CHAPTER 3
Feature-based line segmentation

After completion of the preprocessing steps exposed in Chapter 2, the document is now represented in a binary image, with black foreground handwritten patterns on a white background. Two metrics about the page layout have been computed from the vertical projection of the thresholded image: the top and bottom boundaries of the text block and the average line spacing, or period. These preliminary computations make it possible to devise a bottom-up line detection method.

3.1 An overview

The segmentation process follows a bottom-up approach. It extracts certain local morphological features from the written material and uses them to:

- detect the center axis of each handwritten line;

- aggregate graphical components about the axes, by use of script specific heuristics

Feature extraction The first phase of the treatment isolates morphological attributes of the written page. Connected component analysis (section 3.2.1) attributes distinct labels to non-touching foreground patterns. Even if the resulting map is not a foundation of the line detection process, it provides convenient access to each graphical unit in the page, eliminating the need to scan the entire image in some further treatments. Then a thinning process (section 3.2.2) erodes the patterns until each of them is reduced to a skeleton 1-
pixel thick. The criteria for choosing among a proliferation of thinning algorithms deserves a discussion, in which a handful of parallel thinning algorithms are evaluated. Finally, skeletons are split into their elementary parts: pixel paths or segments (section 3.2.3.2), and path-connecting pixels, or nodes (section 3.2.3.1).

**Segmentation**  The second phase is the segmentation itself. Using a line spacing estimate computed through Fourier analysis (section 2.3), an iterative procedure aggregates the nodes into elongated clusters, whose medial axis matches the hypothetical centerline of the handwritten line (section 3.3.1). The nodes’ cluster membership is the criterion for distributing the segments among the lines. In particular, those segments that cross between two clusters (later referred as bridge-segments) are assigned according to script-specific heuristics (section 3.3.2).

Figure 3.1 on page 45 summarizes the processing pipeline.

### 3.2 Describing the handwritten material

#### 3.2.1 Connected component analysis

A crucial step in many line segmentation procedures (see section 1.3.6.3), the connected component analysis labels those sets of foreground pixels that have the same value, thus detecting every separate graphical unit in the page. As we are about to see, the nature of our documents makes it difficult to use the CC analysis as a foundation for a segmentation procedure. However, it can be very useful for a number of auxiliary procedures, such as noise removal.

##### 3.2.1.1 Component labeling

**Dual adjacency model**  In the binary document image, foreground pixels can be 4-adjacent (connected by one of their side) or 8-adjacent (connected also by their corners). Since all 4-adjacent pixels are also 8-adjacent, the grid foreground is defined by an 8-adjacency model
Figure 3.1: Feature-based segmentation: overview
Figure 3.2: Dual adjacency model: background (white) pixels contained in the loop are not adjacent to other background pixels.

(see Klette and Rosenfeld [40]). The grid background is defined by a 4-adjacency model, so that corner connected background pixels are not adjacent (see Figure 3.2). A set of foreground (background) pixels $S$ is called an 8-connected (4-connected) component if a path of 8-adjacent (4-adjacent) foreground (background) pixels exists between any two pixels of the set.

**Depth-first component labeling**  Connected components can be labeled with procedure 3.1, which applies a depth-first strategy to visit all the pixels in a component. The procedure is then applied to the next unlabeled pixel in the image.

**Algorithm 3.1** Connected component labeling (Klette & Rosenfeld, 2004)

```plaintext
1: function FILL($p$, $label_k$)
2:     Label pixel $p$ with $label_k$
3:     Put $p$ into a stack
4:     repeat
5:         Pop pixel $p$ out of the stack
6:         Label with $label_k$ all unlabeled, foreground neighbors of $p$ and put them into the stack
7:     until stack is empty
```

**3.2.1.2 Using the CC analysis**

In some cases, component labeling might work as a reasonable approximation for the intended segmentation task if the components match easily manageable units such as characters, or words. On the test material chosen for this study, however, CC identification
Figure 3.3: Connected components map (Ms. HRC leaf M3, f96r): the connected component at the center extends over 6 lines.

does little to ease the subsequent segmentation task. Figure 3.3, which shows the result of the procedure above, suggests that CCs are not to be relied on for a segmentation strategy:

- many connected components encompass 2 lines or more; the largest one (Figure 3.3, center) extends over 6 lines;

- on a small page featuring narrow columns, each line contains only a few components; this small population makes it difficult to detect alignments or paths that could be used as descriptors for the centerline;

Basic connecting components metrics are easily extracted by measuring the size of their bounding boxes. The fact that the median height of the components’ bounding boxes (Figure 3.4(a)) is equal to the average line spacing (as measured in section 2.3) is certainly not fortuitous. However, caution should prevent us from generalizing and using the bounding box median height to estimate the line spacing in other documents: the ratio of character height to line spacing may differ significantly from one manuscript to another. The component area distribution, however, can help identify those units that can be considered noise: we can dispose safely of these unit-size entities that produce the peak on the
Figure 3.4: Connected components statistics: (a) The median height (20) is exactly half the average line spacing measured in section 2.3 (b) The peak on the left is made primarily of 1- or 2-pixel sized components.

Even if the connected components are not chosen as a foundation for a bottom-up line detection procedure, they serve as a catalog of foreground patterns, that can easily be accessed and manipulated when needed. For instance, Section 3.3.2.2 uses some connected component metrics to help assign some bridge-segments. The component labeling is all the more useful because the relationship between connected components remains unchanged after completion of the next operation: the thinning procedure.

3.2.2 Thinning

3.2.2.1 Thinning preserves the pattern topology

A thinning procedure shrinks each foreground connected component of the manuscript image into a 1-pixel thick connected skeleton. This is usually done iteratively: each pass deletes those contour pixels that can be deleted without affecting the relation between foreground and background connected components, thus making some of the inner pixels the
**Figure 3.5:** Thinning: obtaining centrally located arcs through pixel erosion: each pass (in gray levels) deletes some pixels from the contour

new contour pixels, to be examined for deletion in the next pass. As shown in Figure 3.5, gradual contour erosion results in centrally located arcs and curves.

Indeed, a thinning process preserves the topological properties of the input binarized image: there is a 1-to-1 relation between the foreground ($F$) connected components of the input image and the foreground connected components of the output image and between the background ($\bar{F}$) connected components of the input image and the $F$-connected components of the output image. In a 8-adjacency grid, such as defined for our binarized manuscript, each 8-connected set of $F$-pixels in the input grid matches an 8-connected skeleton on the output grid; conversely, each 4-connected set of $\bar{F}$-pixels in the input grid matches an 4-connected $\bar{F}$-component on the output grid (see Figure 3.6).

A key concept underlying the design of the topology-preserving thinning algorithm is the notion of *simple pixel* (see Rosenfeld [41]). A pixel $P$ belonging to the $F$-pattern is simple if:

- $P$ is a border point, i.e. 4-adjacent to at least one $\bar{F}$-point;
(a) Input image: 2 background components (the loop of the "p" is one of them); 2 foreground components.

(b) Output image: the topology is not changed (the loop of the "p" is still a separate component).

Figure 3.6: Thinning as a 1-to-1 mapping between connected components (Ms HRC leaf 3, f86r - detail)

- the neighbors of $P$ that belong to the pattern are connected.

Otherwise stated, a simple pixel $P$ can be deleted from the pattern without changing its topology. Moreover, as shown by Ronse [42] in a formal proof, a set of pixels $D$ can be characterized as deletable if pixels of $D$ can be labeled $P_1, P_2, ...$ in such a way that each $P_i$ is simple after $P_{i-1}$ has been deleted from $D$. Since simpleness is a property local to the neighborhood of $P$, a procedure relying on eroding masks can maintain the pattern topology.

That said, a thinning algorithm that breaks the topology of specific, "pathological" patterns (see below) might still provide usable output for the purpose at hand, especially when working with handwritten material. Zhang and Suen’s algorithm [43] is one of those algorithms that provide a very usable, albeit topologically unsound, output. A detailed exposition of its workings and limitations can be found in Appendix C. However, smooth, easily recognizable thinned characters do not need to exist at the expense of topological rigor, as illustrated by the following procedures:

- a 4-pass algorithm (Algorithm 3.2): although it uses Zhang’s generic structure, its
stronger directional constraints preserve the pattern topology

- a 2-pass version of the preceding

- 2 examples of parallel algorithms taken from the literature (Chin, Wan, Stover, and Iverson [44]; Bernard and Manzanera [45]), that rely on deletion and restoring masks

We used it as a basis for our own thinning procedure, which results in clean and topologically sound outputs.

3.2.2.2 A 4-pass thinning algorithm

Basic notions Before discussing Algorithm 3.3 more in depth, let us define the following notation:

- \( F \)-pixels and \( F \)-components are foreground pixels and foreground connected components; \( \bar{F} \)-pixels and \( \bar{F} \)-components are background pixels and background connected components;

- in a \( 3 \times 3 \) sliding mask, \( P_1 \) is the center pixel under consideration and \( P_2, P_3, \ldots, P_9 \) its neighbors, whose respective locations are as follows:

<table>
<thead>
<tr>
<th>( P_9 )</th>
<th>( P_2 )</th>
<th>( P_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_8 )</td>
<td>( P_1 )</td>
<td>( P_4 )</td>
</tr>
<tr>
<td>( P_7 )</td>
<td>( P_6 )</td>
<td>( P_5 )</td>
</tr>
</tbody>
</table>

- when \( P_1 \) is 4-adjacent to the background component by one side only (i.e. when it has 5 or 6 connected neighbors), it is embedded (see Figure 3.7-(a and b)) ; if \( P_1 \) has exactly 3 neighbors, all located in the same row or column, it is an "ear"-pixel (see Figure 3.7-c):
Algorithm 3.2 Four-pass thinning algorithm

Require: \( Img \) a binarized image

1: \textbf{function} \textsc{Thin-4-Pass}(\( Img \))
2: \hspace{1em} \textbf{repeat}
3: \hspace{2em} \textit{changed} \leftarrow \text{false}
4: \hspace{2em} \textbf{for} pass \leftarrow 1 \text{ to } 4 \textbf{ do}
5: \hspace{3em} \textbf{for} every foreground pixel \( P_1 \) \textbf{ in } \( Img \) \textbf{ do}
6: \hspace{4em} \( C_1 \leftarrow \) (card(neighbors of \( P_1 \)) \leq 6)
7: \hspace{4em} \( C_2 \leftarrow \) (card(neighbors of \( P_1 \)) \geq 2)
8: \hspace{4em} \( C_3 \leftarrow \) (card(transitions 0 \rightarrow 1 in sequence \( (P_2, ..., P_9, P_2) \)) = 1)
9: \hspace{3em} \textbf{if} pass = 1 \textbf{ then}
10: \hspace{4em} \( C_4 \leftarrow ((P_2 = 0 \text{ AND } P_6 = 1) \)
11: \hspace{3em} \textbf{else if} pass = 2 \textbf{ then}
12: \hspace{4em} \( C_4 \leftarrow (P_4 = 0 \text{ AND } P_8 = 1) \)
13: \hspace{3em} \textbf{else if} pass = 3 \textbf{ then}
14: \hspace{4em} \( C_4 \leftarrow (P_6 = 0 \text{ AND } P_2 = 1) \)
15: \hspace{3em} \textbf{else if} pass = 4 \textbf{ then}
16: \hspace{4em} \( C_4 \leftarrow (P_8 = 0 \text{ AND } P_4 = 1) \)
17: \hspace{3em} \textbf{else}
18: \hspace{4em} \( C_4 \leftarrow \text{false} \)
19: \hspace{3em} \textbf{if} \((C_1 \text{ AND } C_2 \text{ AND } C_3 \text{ AND } C_4)\) \textbf{ then}
20: \hspace{4em} Mark \( P_1 \) \textbf{ for deletion}
21: \hspace{3em} \textit{changed} \leftarrow \text{true}
22: \hspace{3em} \textbf{for} every foreground pixel \( P \) \textbf{ in } \( Img \) \textbf{ do}
23: \hspace{4em} \textbf{if} \( P \) marked \textbf{ for deletion} \textbf{ then}
24: \hspace{5em} \( P \leftarrow 0 \) \hspace{1em} \( \triangleright \) \text{Delete the pixel}
25: \hspace{2em} \textbf{until} \textit{changed} = \text{false}
26: \hspace{2em} \textbf{for} every foreground pixel \( P_1 \) \textbf{ in } \( Img \) \hspace{1em} \( \triangleright \) \text{Cleans up 4-connections}
27: \hspace{3em} \( C_1' \leftarrow ((P_6 \ast P_8 = 1) \text{ AND } (P_7 + P_3 = 0)) \)
28: \hspace{3em} \( C_2' \leftarrow ((P_8 \ast P_2 = 1) \text{ AND } (P_9 + P_5 = 0)) \)
29: \hspace{3em} \( C_3' \leftarrow ((P_2 \ast P_4 = 1) \text{ AND } (P_3 + P_7 = 0)) \)
30: \hspace{3em} \( C_4' \leftarrow ((P_4 \ast P_6 = 1) \text{ AND } (P_5 + P_9 = 0)) \)
31: \hspace{3em} \textbf{if} \( C_1' \text{ OR } C_2' \text{ OR } C_3' \text{ OR } C_4' \) \textbf{ then}
32: \hspace{4em} \( P_1 \leftarrow 0 \) \hspace{1em} \( \triangleright \) \text{Delete the pixel}
Figure 3.7: Two types of border pixels: (a) and (b) Embedded pixels (c) "ear" pixels

Figure 3.8: A few 3 \times 3 contexts for non-deletable pixels: (a)–(b) create new background components; (c) does not delete an outermost pixel first, thus modifying the pattern morphology; (d)–(f) create new foreground components; (g)–(h) erode some segment ends.

**Permanent conditions for deletion** Algorithm 3.3 retains the general structure and the connectedness conditions $C_1$, $C_2$, and $C_3$ of Algorithm C.10.

In the marking operation, connectedness properties are evaluated in the 3 \times 3 context $N_8(P)$. Although disconnected $\bar{F}$-components or $F$-components in the mask might be part of the same connected component in a larger window, preserving the topology $N_8(P)$ is necessary to maintain the image’s most relevant topological features. Condition $C_1$ (l. 6) states that $P_1$ is connected to at least one $\bar{F}$-component, so that its deletion will not result in a "hole" in the $F$-pattern (see Figure 3.8-(a,b) for an example). Otherwise stated, $P$ is a $F$ border pixel. Condition $C_3$ (l. 8) ensures that $P$ is adjacent to at most one $\bar{F}$-component so that its deletion will not merge two $\bar{F}$-components (see Figure 3.8(e)–(f)). Condition $C_2$ (l. 7) avoids the deletion of a segment end (see Figure 3.8(g)–(h)).
**Directional erosion steps** In each iteration, a 4-step marking-and-deletion process erodes successively the North, East, South, and West border pixels. Pass 1 marks all North border pixels (l. 10) and deletes them all at once (l. 24). Figure 3.9 represents the set of "North" patterns considered in pass 1.

Passes 2, 3, and 4 operate on analog, rotated patterns.

The directional constraints for each step are more strict than in Zhang (see Section C). Sub-iterations discriminate better between corner patterns: only 2 kinds can be deleted in a single step. The North-erosion step does not delete SE and SW corners; the East-erosion step does not delete SW and NW corners, etc. As a result, two sub-iterations share only one flavor of corner patterns (see Figure 3.10).

When the main iteration stops, the pattern is composed of 4-connected pixels (see Figure 3.11(b)); a subsequent step applies a mask to all those patterns that can be made 8-connected without modifying the local topology (see Figure 3.11(b)–(c)).
Figure 3.11: 4-pass thinning algorithm: Ms HRC leaf M3, f86r (detail) – Pattern (b) is produced by the iterative part of the thinning process: connected pixels are 4-adjacent; a 3 × 3 mask slides once more over the pattern to produce 8-paths (c). Pixels removed during this last operation are shown in white in figure (d).

Conditions for directional shrinking  The conditions that guide the deletion process in each pass can be easily inferred from the corresponding set of patterns. Figure 3.9 represents all patterns that result in the suppression of North border pixels. Assuming that $C_1$, $C_2$, and $C_3$ are true (see section 3.2.2.2), the entire set is obtained by the condition $P_2 = 0 \land P_6 = 1$ (l. 10). By analogy, conditions $P_4 = 0 \land P_8 = 1$ (l. 12), $P_6 = 0 \land P_2 = 1$ (l. 14), $P_8 = 0 \land P_4 = 1$ (l. 16) define the East, South and West border patterns respectively.

The mask used to detect 4-adjacent connections on the resulting curves is shown in Figure 3.12, with its $n\pi/2$-rotations. The corresponding conditions $C'_1$ to $C'_4$ (l. 27 to
Figure 3.12: 4-adjacent pixels pattern and its rotations

l. 30) are easily derived. For example, in (a), deleting $P_1$ preserves connectivity between the remaining $F$-cells of the window, excepted $P_3$, whose connection to the pattern would depend on either $P_2$ or $P_4$ being non-empty. Therefore deletion of $P_1$ preserves the cell topology in every case iff $P_3$ is empty.

**Evaluation** Algorithm 3.3 gives satisfactory results on those patterns that cause Zhang’s procedure to fail, as illustrated by Figure 3.13. There is a one-to-one relation between input and output connected components. Diagonal segments undergo a severe compression, but they do not vanish. Even though Lü and Wang’s modification [46] on condition $C_2$ (which increases the minimum number of neighbors to three) solves this particular problem, it causes undesirable noise (spurs) on handwritten patterns, whose edges comprise numerous pixel-size dents and spikes (see Figure 3.14–(a)).

3.2.2.3 A 2-pass thinning algorithm

The four passes of Algorithm 3.2 can easily be regrouped into two, giving Algorithm 3.3. It provides a very similar, albeit non identical, output.

The two-step marking-and-deletion loop erodes alternately the North and East border pixels in pass 1 (l. 9), and the South and West border pixels in pass 2 (l. 11). The directional constraints for each step are stricter than in Zhang’s two-pass algorithm (see Appendix C), because sub-iterations discriminate better between corner patterns: only 3 kinds can be deleted in a single step. The N+E border erosion step does not delete SW corners, the S+W boundary erosion step does not delete NE corners; as a result, the two
Figure 3.13: 4-pass thinning algorithm: difficult cases

(a) Input
(b) Erosion phases
(c) Neighbors ≥ 2
(d) Neighbors ≥ 3 (Lü)

Figure 3.14: Noise in thinned characters.
Algorithm 3.3 Two-pass thinning algorithm

Require: Im a binarized image

1: function THIN-2-PASS(Im)
2: repeat
3: changed ← false
4: for pass ← 1 to 2 do
5: for every foreground pixel $P_1$ in Im do
6: $C_1 \leftarrow (2 \leq \text{card(neighbors of } P_1) \leq 6)$
7: $C_3 \leftarrow (\text{card(transitions } 0 \rightarrow 1 \text{ in sequence } (P_2, ..., P_9, P_2)) = 1)$
8: if pass = 1 then
9: $C_4 \leftarrow ((P_2 = 0 \text{ AND } P_6 = 1) \text{ OR } (P_4 = 0 \text{ AND } P_8 = 1))$
10: else if pass = 2 then
11: $C_4 \leftarrow (P_6 = 0 \text{ AND } P_2 = 1)\text{ OR } (P_8 = 0 \text{ AND } P_4 = 1))$
12: else
13: $C_4 \leftarrow$ false
14: if ($C_1$ AND $C_2$ AND $C_3$ AND $C_4$) then
15: Mark $P_1$ for deletion
16: changed ← true
17: Delete all marked pixels $\triangleright$ See Algorithm 3.2, l. 24
18: until changed = false
19: Process 4-adjacent pixels $\triangleright$ See Algorithm 3.2, l. 32

sub-iterations share only two flavors of corner patterns: the NW and SE corners.

Evaluation The results provided by this 2-pass Algorithm are very similar to those given by its 4-pass flavor, as illustrated by Figure 3.15. Algorithm 3.3 gives passable outcome on the test patterns (Figure 3.16): topology is preserved, even if the segments retract into single pixels. Increasing the minimal number of neighbors for deletable pixels to 3 results in a much noisier shape than for Algorithm 3.2, as shown in Figure 3.14–(b).

3.2.2.4 Complexity

Let $C_1$ be the time taken by the algorithm to check the deletion conditions in the pixel neighborhood and label the pixel accordingly. It is a constant, although it depends on various parameters and can vary significantly with the size of the mask, the number of
Figure 3.15: 2-pass thinning algorithm: Ms HRC leaf M3, f86r (detail)

(a) Ms. HRC leaf 3, f86r
(b) Erosion steps

Figure 3.16: 2-pass thinning algorithm: difficult cases

(a) (b) (c) (d)
conditions and the optimization of the boolean formulas. Therefore the time $t_p$ taken by each mark-and-delete pass on an $m \times n$ image is bounded above by

$$ t \leq mnC_1 \quad \text{for checking deletion conditions on } m \times n \text{ pixels} \quad (3.1) $$

$$ + \quad mn \quad \text{for deleting marked pixels} \quad (3.2) $$

$$ = \quad (C_1 + 1)mn \quad (3.3) $$

$$ = \quad C_2mn \quad (3.4) $$

Therefore, if $p$ is the number of sub-iterations ($p = 4$ in Algorithm 3.2, for instance), each loop runs in $p \times t_p = pC_2mn = C_3mn$ time.

The number of iterations needed to erode a pattern of size $m \times n$ is $m/2$ (if $m \leq n$) or $n/2$ (if $n \leq m$), as illustrated by Figure 3.17, in which each iteration erodes one layer of border pixels off the square pattern. The times $T_1$ and $T_2$ taken by, respectively, Algorithm 3.2 (4-pass) and Algorithm 3.3 (2-pass) to process an image of size $n$ are

$$ T_1(n) = Cmn^2 \in O(mn^2) \quad (3.5) $$

and

$$ T_2(mn) = 2T_1(mn) \in O(mn^2) \quad (3.6) $$

In practice, the number of iterations required to thin a handwritten pattern is much smaller, since the proportion of border pixels in stroke-based shapes is usually higher. With a number of deletion phases closer to $\log(n)$ than $n/2$, the thinning completion time is contained in $O(mn\log(n))$. For instance, thinning our 1750 $\times$ 575 pixel test page (Ms. HRC leaf M3, f96r, left column), whose foreground pixels cover about 40% of the grid, takes about 10 to 12 iterations, depending on the algorithm used. Thus optimization should focus on minimizing the number of operations performed in each iteration, instead, in order to reduce the constant factor $C$. For clarity purpose, the outlines of Algorithm 3.2 and 3.3 do not include obvious steps that can shave significant time off every iteration, such as sliding
the mask as soon as one of the condition fails. A fully parallel implementation, in which boolean operations are performed on the entire image by a dedicated cellular automaton performing a reduced set of combinatorial operations (as in Manzanera, Bernard, Prêteux, and Longuet [47]), is another way to improve the efficiency of these algorithms.

As a matter of fact, the two algorithms that we are about to present were developed with a parallel-processing framework in mind. They are nonetheless of interest for this project, since they introduce the concept of restoring mask, which is not explicit the procedures exposed so far.

### 3.2.2.5 Two other algorithms

Out of a plethora of 2D thinning algorithms,\(^{13}\) Algorithms 3.4 and 3.5 share the same structure. Both procedures use only a single pass and preserve topology.\(^{14}\)

**Chin & Wan (1987)** Chin, Wan, Stover, and Iverson [44] design a fully parallel algorithm: the image is checked for simple pixels in a single pass; all marked pixels are then deleted at once (Algorithm 3.4, l. 3). The iterations stop when the system stabilizes. The authors apply two different thinning masks (see Figure 3.18(a)–(b)) to detect candidate

\(^{13}\)More than 100: see Klette and Rosenfeld [40] for an extensive, albeit non exhaustive, list.

\(^{14}\)As shown by Couprie [48], who provides an experimental confirmation of the detailed proof given by Bernard and Manzanera [45].
Figure 3.18: Thinning and restoring masks for Chin and Wang’s algorithm: (a)–(b): thinning masks (c)–(d): restoring masks

simple pixels in the pattern. Both masks check for a pattern in $N_8(P)$, through every $\pi/2$-rotation. $P$ is marked for deletion iff $N_8(P)$ matches any one of those masks and does not match any of the restoring masks (c) and (d). Since there is only one deletion pass in each iteration, exploring $N_8(P)$ only does not ensure that $P$ is deletable. Hence the distinctive shape of the restoring masks, which extend beyond $N_8$, in the horizontal and vertical axes, respectively.

Algorithm 3.4 One-pass parallel thinning algorithm (Chin & Wan, 1987)

1: repeat
2:   $Y \leftarrow$ set of pixels in $F$ matching masks (a) or (b) but none of (c) and (d)
3:   $X \leftarrow X \setminus Y$
4: until $Y = \emptyset$

Figure 3.19 illustrates a few steps of Chin’s procedure running on our favorite manuscript detail (the bar above the $\bar{p}$ in “$\bar{p}d\bar{i}$”): since pixel (9,36) matches the thinning mask (a) but not the restoring mask (d), it is not marked for deletion, ensuring the connectivity of the pattern. Thanks to the small, rotating masks that slide in a single pass, the erosion is more symmetrical than in Algorithm 3.2 and 3.3. Order dependency shows only on 2-pixel thick vertical and horizontal segments, which are eroded from South and East, respectively. The restoring masks are indeed directional.

As suggested by Figure 3.20, this algorithm gives very good results on pathological patterns (squares and 2-pixels thick segments), at the expense of noise on those handwritten patterns whose borders have dents and spikes. This, as well as our experiments with Algorithm 3.3 (Figure 3.14(b)), illustrates a difficulty common to all thinning algorithms,
Figure 3.19: Delete and restore masks (Chin & Wan’s algorithm, 1987)
Figure 3.20: Chin & Wan’s thinning algorithm (1987)

Figure 3.21: Thinning and restoring masks for Bernard and Manzanera’s algorithm: (a) and (b) rotating thinning masks (c) rotating restoring mask

whose requirements: medial axis location, topology preservation, one-pixel thickness, and noise immunity cannot be satisfied equally well.

Bernard & Manzanera (1999)  The algorithm BM2, exposed in Bernard and Manzanera [45], is an evolution of the BM algorithm by the same team (see Manzanera, Bernard, Prêteux, and Longuet [47]). As in Chin and Wan’s algorithm [44], the single-pass procedure relies on 2 thinning masks and a restoring pattern. Those pixels that match either one of the first two, and not the latter, are marked for deletion. Then all marked border pixels are suppressed at once. BM2 algorithm's originality lies in the use of larger thinning masks. In all their rotations, they cover the $5 \times 5$ neighborhood of the pixel under consideration. In order to obtain a homotopic (or topology-preserving) and centrally located skeleton, the authors deliberately relax the 1-pixel thickness constraint and use a symmetric, rotating
Figure 3.22: BM2 thinning algorithm (1999)

restoring mask.

**Algorithm 3.5** Fully parallel thinning algorithm (Bernard & Manzaneras, 1999)

1: repeat
2: \( Y \leftarrow \) set of pixels in \( F \) which match thinning masks (a) or (b) but not (c)
3: \( X \leftarrow X \setminus Y \)
4: until \( Y = \emptyset \)

Assuming that the 1-pixel thickness is not a mandatory requirement, the output of the BM2 algorithm compares favorably with the results of Chin’s procedures: character morphology is very well preserved, with a skeleton located on the medial axis of every limb. The algorithm looks relatively immune to noise.

**Combining algorithms** Figure 3.23 features a visual summary of this short survey of thinning techniques. In order to extract the character features most useful for the line detection phase, the output thinning algorithm must satisfy the following requirements, in that order:

1. the skeleton is 1-pixel thick
2. the skeleton preserves the stroke axis
3. noise is minimal
Figure 3.23: Skeletons: comparison
Figure 3.24: Segments and nodes

Two other properties: strict homotopy and mediality (i.e. overlapping between the segments and the pattern medial axis) are not critical to our purpose. From the experiment above, it appears that the best way to obtain (1) and (3) is to combine two thinning algorithms: BM2 is run first over the pattern, providing a 2-pixel thick pattern satisfying (2) and (3), as well as the desirable, albeit optional, properties of topology-preservation and mediality; our 4-pass procedure then shrinks the pattern to its final form (Figure 3.23(e)).

The result on the entire handwritten page can be see in Appendix A.5.

3.2.3 Identifying segments and nodes

From the thinned connected components it is now possible to extract salient features through straightforward procedures. Each skeleton is made of paths 1-pixel thick, which match the manuscript pen strokes. We referred to them as segments. Segments are connected to each other through node pixels, as seen on Figure 3.24. For practical matter, nodes are identified first, then the segments are retrieved from the nodes.

3.2.3.1 Nodes

**Definition 1** A pixel is a node if an only if it is adjacent to at least 3 background connected components in the $3 \times 3$ neighborhood.

This definition allows us to retrieve any segment intersection pattern (Figure 3.25-(a)-(b)), including adjacent nodes (Figure 3.25-(d)). Background adjacency is easily tested by counting the $0 \rightarrow 1$ transitions in $P$’s ordered set of neighbors (see Algorithm 3.6, l. 3).
Figure 3.25: Node detection patterns: (a) N has 3 neighbors, each of them belonging to a different segment (b) N connects 4 segments (c) N has 3 neighbors, but is not a node (d) An instance of adjacent nodes: each of $N_1$ and $N_2$ is adjacent to 3 background connected components only.

Algorithm 3.6 Node identification procedure

Require: $Img$ a binarized image, with thin foreground patterns

1: function FINDNODES($Img$)
2: for every foreground pixel $P$ in $Img$ do
3: \[ C \leftarrow \text{card(transitions } 0 \rightarrow 1 \text{ in sequence } (P_1, ..., P_8, P_1) \geq 3) \]
4: if $C = \text{true}$ then
5: Label $P$ as a node

As shown by Figure 3.26, the nodes can be useful descriptors of the handwritten line:

- because they usually match the location where pen strokes intersect, most are close to the line horizontal axis; for the same reason, nodes give a description of the line where accents and subscript shorthand marks have much less weight: because they are made of a single stroke most of them do not include nodes.

- the short lines of our test material rarely contain more than 5 or 6 connected components, but at least 3 times as many nodes

- nodes allow us to retrieve structural components of the handwriting: the segments.

3.2.3.2 Segments

Segments are retrieved through a recursive search on each node’s non-zero neighbor. Algorithm 3.7 gives a broad outline of the procedure, which returns the segment as a vector
Figure 3.26: Manuscript HRC_leaf_M3, f96r: Nodes

of pixels. A defined search order (l. 3) for neighborhood exploration takes advantage of the thinned pattern properties to speed up the search and avoid some possible pitfalls, such as the "8-adjacent neighbor trap", illustrated by Figure 3.25(a). Indeed, our particular thinning procedure results in a thin pattern property that can be expressed in 2 equivalent ways:

1. if two neighbors of a non-node pixel $P$ are mutually adjacent, then the 4-adjacent neighbor is a node pixel

2. a non-node pixel cannot have two mutually adjacent non-node neighbors

Formula (1) implies that the 4-adjacent neighbor $N_P$ of a non-node pixel $P$ should be searched first, as illustrated in Figure 3.27(b)–(c). If $N_P$ is a node, this priority ensures that the search ends properly, avoiding the construction of a super-segment (composed of two segments, that would otherwise be retrieved separately) or, in the case of a self-closing segment, the infinite loop shown in Figure 3.27(a). Formula (2) makes clear that a search order, albeit not critical for pixels that are not adjacent to a node-pixel, can make the process
quicker, by restricting the set of visitable neighbors.

**Algorithm 3.7** Segment retrieving procedure

**Require:** Img is a binarized image, with thin foreground patterns

1: function RETRIEVESEGMENT(Img, P<sub>x</sub>,y, searchOrder)
2: if is.EndPixel(P<sub>x</sub>,y) or is.NodePixel(P<sub>x</sub>,y) then return P<sub>x</sub>,y
3: for every neighbor P<sub>n</sub> of P<sub>x</sub>,y in the order given by searchOrder do
4:    nextSearchOrder ← COMPUTESEARCHORDER(P<sub>n</sub>)
5: return Concat(P<sub>x</sub>,y, RETRIEVESEGMENT(Img, P<sub>n</sub>, nextSearchOrder))

We call **internal** any segment that is a path between two nodes. Those nodes may be distinct, or not: thus internal segments include loops. We call **external** those segments which have a free end and connect to exactly one node. Figure 3.28 shows the result of the segment retrieval procedure, with a distinct color for the internal segments. As approximations of the pen strokes, segments yield useful information about the handwritten signs: metrics (length, curvature) can be computed easily, for instance. In the perspective of this work, segment intersections provide good candidates for locations to cut between lines, as exposed in the coming section.

### 3.3 Clustering-based segmentation

The line segmentation procedure is performed in 3 steps:
1. the average line spacing over the page is computed; this statistic is crucial for the clustering step;

2. nodes are classified in as many groups as there are lines in the page: a bottom up clustering algorithm groups the nodes along hypothetical centerlines, line by line, until the entire page has been processed (Figure 3.29(a));

3. then each segment is assigned to a line, depending on its end node’s cluster membership. Segments that connect nodes belonging to 2 different clusters (i.e. bridge-segments) (Figure 3.29(b)) require closer examination: in these cases, line assignment decision is guided by script-specific heuristics.

### 3.3.1 Clustering the nodes

The node clustering procedure described in this section assumes that the line spacing average, a crucial measure, has been computed in a prior step, by computing the power
Figure 3.29: Overview of the segmentation procedure: (a) Clustering the nodes to detect the lines (b) Identifying bridge-segments and potential cut points.

The node clustering procedure is inspired by the least square clustering technique known as the $k$-means method (see A.K. Jain and Flynn [49] or Manning and Schütze [50] for a detailed presentation). This method, which defines clusters by the center of mass of their members, works as follows:

1: function $k$-MEANSCLUSTERS(objects)
2: Define initial cluster centers
3: repeat
4: for each object $obj$ do
5: Assign $obj$ to the center that is closest
6: for each cluster do
7: Recompute the center as the mean of its members
8: until Clusters stabilize

Figure 3.30 illustrates one iteration of the $k$-means method, in which the recomputation of the center coordinates typically relies on the Euclidean distance and the unweighted mean function.
**Figure 3.30:** An iteration of the $k$-means clustering process (from Manning & Schütze, 1999).

**Figure 3.31:** Recomputing a cluster centerline with a least-square fit

In practice, any distance or mean function may be used, depending on the task at hand. While retaining the core principle of the $k$-means method, the clustering procedure presented in Algorithm 3.8 (p. 74) is slightly more complex. The overall structure is the same: the procedure iterates (outer loop, l. 6–29) until no change occurs in the node cluster assignments. However, in order to adapt the $k$-means method to line detection, substantial modifications are introduced.

**Elongated clusters** Since the clusters are to be defined by their medial axis, Algorithm 3.8 uses a least square linear fit, instead of a mean function. Figure 3.31 shows how a cluster centerline temporary estimate is recomputed to fit the cluster members. Centerline recomputation occurs as soon as a cluster has been assigned all possible nodes for the current iteration (l. 20).
Algorithm 3.8 Node clustering algorithm

Require: Period, i.e. the line period in the page, has been computed in a prior step.

1: function NodeClusters(nodes)
2:     iteration ← 1
3:     lineNumber ← 1
4:     Estimate Line_N = A_N x + B_N \quad \triangleright \text{A crude estimate of the top line axis}
5:     Cluster_N ← EmptyCluster()
6:     Cluster_S ← EmptyCluster()
7:     repeat
8:         lineNumber ← lineNumber + 1
9:         repeat
10:            if iteration = 1 then \quad \triangleright \text{Estimate the next line from the current one}
11:                Line_N ← A_N x + B_N
12:                Line_S ← A_N + (B_N + Period)
13:            else \quad \triangleright \text{Use the lines computed during the first iteration}
14:                Line_N ← storedLines[lineNumber]
15:                Line_S ← storedLines[lineNumber + 1]
16:         for each node n located between Line_N and Line_S do
17:            if distance(n, L_N) ≤ distance(n, L_S) then
18:                add n to Cluster_N
19:            else
20:                add n to Cluster_S
21:       A_N x + B_N(\cdot) ← least square fit of Cluster_N \quad \triangleright \text{Cluster}_N \text{ is now complete}
22:       storedLines[lineNumber] ← A_N x + B_N(\cdot)
23:       Cluster_N ← Cluster_S
24:       Cluster_S ← EmptyCluster()
25:       lineNumber ← EmptyCluster()
26:       if iteration = 1 then
27:           B_N ← B_N + Period
28:       until End of page reached
29:       iteration ← iteration + 1
30:     until Clusters stabilize

Incremental cluster creation \quad \text{In the } k\text{-means method, the number of clusters to be populated is known from the start; instead, Algorithm 3.8 makes no such assumption. Starting from the top, an iterative procedure (inner loop, l. 8) scans the page and builds the clusters incrementally, never considering more than 2 clusters at a time or, in more accurate terms, more than 2 half-clusters, as shown by Figure 3.32 (p. 75). Points located between the line estimates North and South are distributed according to their distance to each line}
Figure 3.32: Assigning nodes to North and South half-clusters: nodes in gray are not considered for assignment; gray North-labeled nodes have been assigned in the preceding step.

(Algorithm 3.8, p. 74, l. 16–19). When all nodes have been assigned, the North cluster is complete, since it has been assigned some of the nodes above in the preceding step. However, the South cluster will not be completed until the next step, at which point it becomes the new North cluster (l. 22).

**Temporary centerline estimation with the period** Assuming that variations in line spacing over the page are contained in a relatively narrow range, the statistic obtained in section 2.3 is sufficient to provide a reasonable approximation of the South cluster centerline at each step of the algorithm: assuming that the North center line $y = Ax + B$ is known from the precedent step, the estimate for the South center line has the same slope and its intercept is $B + \text{Period}$ (l. 11). After the first 10 lines, the Fourier estimate is replaced by a running average of the last 10 interlinear spacings: it has been found to significantly improve the quality of the clustering when the line-to-line spacing varies through the page.

**Estimating the first line** At the very start of the procedure, when the first iteration starts, an initial estimate of the first line is necessary (l. 3). Since an approximate location for the column top boundary has been computed from the thresholded vertical projection (see Section 2.2.3), we use it to bootstrap the clustering procedure. The horizontal line corre-
Figure 3.33: Clustering algorithm – first two line estimates: line $\hat{L}_1$ is the initial estimate, given by an heuristic; line $\hat{L}_2$ is estimated from $\hat{L}_1$: same slope, intercept at $\hat{L}_1 + \text{Period}$.

Corresponding to the top block boundary is first shifted South by a small amount, closer to the core strip: the correction is equal to the core strip height (as estimated by the median connected component height). Then all nodes that are located at a short distance ($\text{Period}/2$) from this horizontal line are aggregated into a cluster. The line is then adjusted to fit the cluster, thus becoming the first line axis. This crude estimate is sufficient to build a reasonable approximation of the first North cluster.

Detecting the end of page The bottom block boundary computed in Section 2.2.3 is a straightforward means for detecting the end of the page. As soon as it has been reached, the first iteration of the clustering algorithm stops. The value of the line counter at this point is the line count for the column. In all subsequent iterations, the line number value is used to detect the end of page.

An illustration Figures 3.33–3.37 illustrate the first two inner iterations of Algorithm 3.8, starting from the top of the page. The first line $\hat{L}_1$ (Figure 3.33) is the rough initial estimate of the first cluster medial axis (Alg. 3.8, l. 3): this educated guess, based on the text block top boundary estimate, undergoes significant corrections in the following step. The
Figure 3.34: Clustering algorithm – node assignment: for nodes between the two line estimates, distance to each line is computed, allowing for their distribution in clusters 1 and 2; nodes located North of the initial line are included.

Figure 3.35: Clustering algorithm: recomputing center line for North cluster: a new line $L_1$ is computed, that fits the nodes in cluster 1.

Second line $\hat{L}_2$ is itself estimated from estimate $\hat{L}_1$, i.e. a translation by the line period $P$ (Alg. 3.8, l. 11). Then (Figure 3.34) all nodes located between $\hat{L}_1$ and $\hat{L}_2$ are assigned to their respective clusters (Alg. 3.8, l. 16–19). Since the North cluster is also the first cluster in the page, nodes located North of $\hat{L}_1$ are also included. When all nodes are assigned, cluster 1 is complete (Figure 3.35): a new center line $L_1$ is computed, that fits its member nodes (Alg. 3.8, l. 20). At the end of the first iteration (Figure 3.36), the line intercept is incremented by $P$ (Alg. 3.8, l. 26), so that the centerline of cluster 2 becomes $\hat{L}_2$, a better
estimate based on the fitted line $L_1$. When iteration 2 starts, $\hat{L}_2$ is the center line of the new North cluster (Alg. 3.8, l. 10). $\hat{L}_3 = \hat{L}_2 + \text{Period}$ is an estimate for the new South cluster line (Alg. 3.8, l. 11). Node assignment is then performed for $\hat{L}_2$ and $\hat{L}_3$ (Figure 3.37), thus completing the population of cluster 2. A new center line is computed for cluster 2.

Figure 3.38 (p. 80) shows the result of the clustering algorithm on our sample manuscript, along with the cluster centerlines. The line skew ranks from $-3.6^\circ$ to $+3^\circ$, with a maximum line-to-line skew difference of $4^\circ$, showing that the reliance on parallel line estimates in the first iteration is not an obstacle to a successful detection of skew variations between lines, provided they are contained in a reasonable range, which can be expected in our document.

The result of the clustering procedure is stored in two structures:

1. a table mapping each node to its cluster;

2. a list of the cluster-fitted axis lines

These data are used in the following step, in order to assign each segment to a cluster.
3.3.2 Assigning the segments to the clusters

3.3.2.1 Bridge-segments

With nodes assigned to the line clusters, it is possible to find those segments which cross the cluster boundary, that is, the segments whose ends belong to two different clusters: we refer to them as bridge-segments.

Internal bridge-segments are easily identified, since both end pixels are nodes: for each internal segment, the end pixels are matched against the cluster map. If their respective clusters are different, the segment is added to the bridge-segments list.

Identifying external bridge-segments requires an extra step. The end pixel that is not a node is first assigned a cluster by looking for the closest cluster axis line. Then the node pixel is found in the cluster map: if the two cluster assignments do not match, the segment is classified as a bridge-segment.

Figure 3.39 (p. 81) shows the result of the procedure above, with the bridge-segments highlighted. Distinctive colors are used for internal bridges (black), north-connected external bridges (orange), and south-connected external bridges (red). A careful examination of
the page leads to the following observations:

- Most internal bridge-segments correspond to descenders of the North line touching an ascender of the South line or an accent (Figure 3.40(a));

- most external bridge-segments that connect to the North cluster correspond to descenders of the North line (Figure 3.40(b)); they do not extend beyond the South centerline;

- most external bridge-segments that connect to the South cluster extend well over the North centerline, since they match an entire character of the North line (typically an initial \( S \) ("\( \int \)) or \( I \), with its descender (see Figure 3.40(c)); other South-connected external segments are part of accents or abbreviation marks (Figure 3.40(d)).
3.3.2.2 Assignment heuristics for bridge-segments

Heuristics can be derived, which result in a correct line assignment in most cases and do not preclude further refinements.

1. internal bridges are assigned to the same cluster as that of their North node

2. North-connected external segments are assigned to the same cluster as that of their North node

3. South-connected external segments are assigned to the same cluster as that of their North pseudo-node

As shown by Figure 3.40(c), rule (2) is necessary to keep the integrity of the North line. Granted that each segment is not divided at this step and is assigned as a whole, assigning South-connected external segments to the South line would otherwise create a gap in the North line, where they extend as full-fledged characters. The reverse is not true for North-connected segments, which match only the lower stroke of North-located characters.
However, rule (2) results in incorrect assignments for South-connected segments that are part of an accent or abbreviation mark located in the upper strip, as becomes clear in Figure 3.41. Rule (2) needs thus to be refined to take care of segments belonging to interlinear marks.

By looking up any random pixel of a South-connected segment in the connected component map (see Section 3.2.1), it is possible to retrieve the graphical unit to which it belongs, along with some useful bounding box metrics. If the enclosing box of the parent component is entirely contained in the upper strip, for instance, the external segment under consideration is certainly part of an abbreviation mark or an accent. Therefore, it has to be assigned to the South cluster. Only a broad delimitation of the upper strip is needed for this task. By defining the upper strip as the central half of the interlinear space at the given location, we obtain a correct line labeling for all similar patterns.

Figure 3.43 shows that some connected components (in gray) have not been assigned to a cluster yet. Indeed, since the procedure exposed in Section 3.2.3.2 used the nodes as starting points for segment construction, it did not detect those segments that do not attach to any node. The following section deals with these remaining entities.

### 3.3.2.3 Assigning free segments

A **free segment** is a segment that does not connect to any node. Assuming that internal and external segments have been recovered in a previous step, free segments can
Figure 3.42: Assignments heuristics for accents: the parent component of the blue segment is entirely contained in the upper strip.

Figure 3.43: Assignment of internal and external segments (Ms. HRC_leaf_M3, f96r): gray-colored connected components are unassigned free segments.
be obtained simply by subtracting all non-free segments from the input image and scanning
the resulting grid for end-pixel patterns, as outlined in Algorithm 3.9. **Rings** or loops are

**Algorithm 3.9** Free segment retrieving procedure

1: function FREESEGMENTS(Img)
2:     for each foreground pixel \( P \) in \( \text{Img} \) do
3:         if \( P \) has only 1 foreground neighbor then
4:             RetrieveSegment(Img, \( P \), SearchOrder) \>
See Algorithm 3.7

also processed at this point: after subtracting the free segments from the image, they are the
only remaining patterns on the page. The ring retrieval procedure uses an arbitrary pixel on
the pattern as a starting point for ring construction.

Distribution of the free segments and rings among the lines does not rely on end
pixels. The center of each segment’s enclosing box is chosen instead as a proxy for cluster
assignment. This choice is motivated by the following facts (see Figure 3.44): with very
few exceptions, free segments are either abbreviation marks and accents located in the
upper strip of a given line, or single characters belonging to the core strip; rings are always
contained in the core strip. Otherwise stated, free segments and rings do not connect two
lines together and can therefore be assigned without considering whether they cross a line boundary or not. Hence the following heuristics:

1. if the segment’s enclosing box is contained in the upper strip (as defined in Section 3.3.2.2), then assign it to the closest line below;

2. otherwise assign the segment (or ring) to the closest line.

Figure 3.45 shows the output of the last segment assignment step (see Appendix A.6 for the complete page). These labeled lines can be used as an input for character segmentation programs, many of which work with thinned patterns. Figure 3.46, which displays the separated skeletons as an overlay, permits a subjective appreciation of the quality of the line segmentation. Restoring the characters to their original shape is a possible and useful extension to the algorithm: this point is discussed in the last part of this study, along with the outcome of the tests.
Figure 3.46: Manuscript HRC_leaf_M3, f96r: line separation
4.1 Sample manuscripts

The characteristics of the manuscripts chosen for the testing set are summarized in Table 4.1. All scripts and layouts were commonly used in the 15th century. As exposed in Section 1, formal scripts are out of consideration. The handful of scripts selected for this test are just a few examples of the many "hands" (scripts) used at this time.\(^{15}\)

The first two manuscripts (HRC leaf M3 [28] and FO Z113.P3 [29] (Appendix B.1)) were used as test input during the conception phase of the segmentation algorithm. With straight or only moderately skewed lines and and no degradation that could be troublesome in the binarization phase, they meet the constraints described in Section 1: their cursive script and their tight line spacing allow for a sound testing of the segmentation process, with only minimal preprocessing treatments.

Codices 1008 [51] (Appendix B.3), 1068 [52] (Appendix B.4), and 1086 [53] (Appendix B.5) are three manuscripts from the Cathedral Library of Cologne. They have many features common to the first two, even though the line spacing is more regular and the script slightly more formal. Codex 1008, in particular, is not written in cursive style. Manuscript HRC 153 [54] (Appendix B.2) is characterized by a different layout, with long, straight lines in a single column. On all these manuscripts, the algorithm was expected to perform well.

\(^{15}\)During the 14th and 15th centuries, the hybridization of formerly well-defined styles gives way to a proliferation of scripts, whose variety defies classification.
<table>
<thead>
<tr>
<th>Id</th>
<th>Ms.</th>
<th>Origin</th>
<th>Preservation</th>
<th>Char. size (mm)</th>
<th>Char. size (pixels)</th>
<th>Line length (glyphs)</th>
<th>Script features</th>
<th>Line features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HRC</td>
<td>leaf M3 (f96)</td>
<td>1425-1450, Germany</td>
<td>Good; dark-brown, homogeneous ink; no stain</td>
<td>2.4</td>
<td>19</td>
<td>19</td>
<td>Cursive style</td>
</tr>
<tr>
<td>2</td>
<td>FO</td>
<td>Z113.P3 (f1)</td>
<td>1425-1450, Germany</td>
<td>Good; light-brown, partially faded ink</td>
<td>2.4</td>
<td>28</td>
<td>20</td>
<td>Cursive style</td>
</tr>
<tr>
<td>3</td>
<td>HRC</td>
<td>153 (f21)</td>
<td>1440-1460, Germany</td>
<td>Good; brown, homogeneous ink</td>
<td>1.9</td>
<td>20</td>
<td>54</td>
<td>Cursive style</td>
</tr>
<tr>
<td>4</td>
<td>Codex</td>
<td>1008 (f2)</td>
<td>1470, Germany</td>
<td>Good; light-brown ink</td>
<td>2.6</td>
<td>16</td>
<td>22</td>
<td>Simplified gothic, with well-formed strokes</td>
</tr>
<tr>
<td>5</td>
<td>Codex</td>
<td>1068 (f2)</td>
<td>1485, Germany</td>
<td>Good; light-brown ink</td>
<td>2</td>
<td>16</td>
<td>34</td>
<td>Cursive style</td>
</tr>
<tr>
<td>6</td>
<td>Codex</td>
<td>1086 (f1)</td>
<td>late 15th, Germany</td>
<td>Good</td>
<td>2</td>
<td>16</td>
<td>22</td>
<td>Cursive style</td>
</tr>
<tr>
<td>7</td>
<td>BANC MS UCB 85 (f82r)</td>
<td>1385-1415, Italy</td>
<td>Average; translucent light-brown ink</td>
<td>2.1</td>
<td>25</td>
<td>45</td>
<td>Gothic cursive</td>
<td>Long, tightly packed lines; slight fluctuations in the lower half of the page.</td>
</tr>
<tr>
<td>8</td>
<td>MSB 23</td>
<td>1400-1450, Italy</td>
<td>Mediocre: stains and holes; partially faded brown ink</td>
<td>2.1</td>
<td>20</td>
<td>49</td>
<td>Loose, much simplified gothic, with well-separated, stick-shaped strokes</td>
<td>Long, irregularly spaced lines; general skew affecting the whole page; strong, localized fluctuations</td>
</tr>
</tbody>
</table>

**Table 4.1:** Testing set
Manuscripts BANC UCB 85, folio 82 [55] (Appendix B.6) and MSB 23 [56] (Appendix B.7) are intended to be more challenging. Both contain long (about 45 glyph long,\textsuperscript{16}) fluctuating lines, as it is typical in single-column layout when the scribe has not previously ruled the support with a lead point. The second manuscript is somewhat degraded; since this thesis’ primary focus is not on the preprocessing treatment, the binarization step clearly falls short of providing an optimal input for the segmentation pipeline. This is an opportunity to evaluate the sensitivity of our algorithm to noise.

As shown by Table 4.1, image definition can vary significantly from one manuscript to another, depending on the scanning parameters. Manuscripts (6) and (7) have 2 mm-sized characters,\textsuperscript{17} for instance, that are 16 pixels high in the first image and 25 pixels high in the second image. Since the segmenting procedure has been designed to be independent from the image resolution, these variations were not expected to affect the performance.

4.2 Test results

4.2.1 Scoring system

Three statistics measure the algorithm performance:

**Line recall** (Tableau 4.2, col. 2) A line is correctly identified if it matches one and only one line in the line axes computed by the clustering procedure. If a computed line axis runs on two handwritten lines, for instance, those two lines are not correctly identified. Any handwritten line matched by two cluster axes is not correctly identified.

**Segment recall** (Tableau 4.2, col. 4) Any mis-assigned segment counts as an error, no matter its size, or its function as a part of a bigger graphical sign (the abbreviation

\textsuperscript{16}The line length is obtained by counting all the letters and abbreviations contained in a line.
\textsuperscript{17}The character size is actually the size of the core strip. It can be computed with precision from the page height, a mandatory entry in any manuscript record. The median of the connected components’ bounding box heights has been found to be a very good approximation for this metric.
Table 4.2: Recall statistics

<table>
<thead>
<tr>
<th>Shelf mark</th>
<th>Lines</th>
<th>Found</th>
<th>Schts.</th>
<th>Errors</th>
<th>Recall (%)</th>
<th>Glyphs</th>
<th>Errors</th>
<th>Recall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 HRC leaf M3</td>
<td>40</td>
<td>40</td>
<td>2470</td>
<td>31</td>
<td>99</td>
<td>743</td>
<td>10</td>
<td>99</td>
</tr>
<tr>
<td>2 FO Z113.P3</td>
<td>47</td>
<td>47</td>
<td>4164</td>
<td>37</td>
<td>99</td>
<td>940</td>
<td>16</td>
<td>99</td>
</tr>
<tr>
<td>3 HRC 153</td>
<td>33</td>
<td>33</td>
<td>8190</td>
<td>20</td>
<td>99</td>
<td>1782</td>
<td>7</td>
<td>99</td>
</tr>
<tr>
<td>4 Codex 1008</td>
<td>41</td>
<td>41</td>
<td>2653</td>
<td>19</td>
<td>99</td>
<td>902</td>
<td>12</td>
<td>99</td>
</tr>
<tr>
<td>5 Codex 1068</td>
<td>44</td>
<td>44</td>
<td>4691</td>
<td>20</td>
<td>99</td>
<td>1496</td>
<td>14</td>
<td>99</td>
</tr>
<tr>
<td>6 Codex 1086</td>
<td>38</td>
<td>38</td>
<td>11055</td>
<td>23</td>
<td>99</td>
<td>836</td>
<td>8</td>
<td>99</td>
</tr>
<tr>
<td>7 MS UCB 85</td>
<td>53</td>
<td>53</td>
<td>17563</td>
<td>88</td>
<td>99</td>
<td>2385</td>
<td>38</td>
<td>98</td>
</tr>
<tr>
<td>8 MSB 23</td>
<td>44</td>
<td>12</td>
<td>5701</td>
<td>-</td>
<td>-</td>
<td>2156</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

bar in "ḥoi" (hominid) is made of 2 segments, for instance). This measure is sensitive
to the quality of the binarization and thinning operations. A suboptimal binarization
procedure usually creates cavities and indented edges that result in numerous loops
and spurs in the thinned page, thus increasing the segment count significantly.

**Glyph recall** (Tableau 4.2, col. 6) No matter how relevant the segments for the inner work-
ing of the segmentation procedure, it is not an appropriate unit to evaluate the pro-
gram from a user’s perspective. The glyphs, i.e the letters and symbols, are more
appropriate to a subjective evaluation of the procedure. For purpose of clarity, a less
than rigorous definition of glyph is used here. A glyph is

- a letter

- a separate abbreviation mark (either a subscript mark or an in-line sign)

- an abbreviation mark associated with a letter (such as crossbars (\(\mathcal{L}\)) or loops
  (\(\hat{f}\))

The glyph recall is measured with the following scoring system: a mislabeled letter
costs 2 points; a mislabeled abbreviation mark or part of a letter costs 1 point; a
mislabeled part of an abbreviation mark costs .5 of a point.
4.2.2 Successful performance of the algorithm

On manuscripts 1-6, which were well within the range of acceptable inputs, the segmentation yields excellent results, with few significant errors: the glyph recall approaches 100%. Moreover, the algorithm deals very well with the narrowly-spaced and fluctuating lines of manuscript 7. The segment-based separation approach appears to be a worthy solution to the problem of line overlapping and line touching, as illustrated by Figures 4.1(a)–(d) (complete segmented pages are in Appendix B). Image resolution does not seem to affect the result.

4.2.3 An example of algorithm failure

The segmenting procedure was not anticipated to perform well on Ms. 8 (MSB 23): line fluctuations are more localized but significantly stronger than in Ms. 7; the script, with its stick-shaped characters, is not the intended input for the procedure. Indeed, the line axes computed by the clustering operation miss most handwritten lines, causing the entire segmentation procedure to fail, as illustrated by Figure 4.2, in p. 94: hence, the absence of segment and glyph recall data in Table 4.2. Two factors explain the failure:

- since the script is made of loose segments, it contains very few nodes in its thinned form, making it harder to form line-matching clusters;

- the fluctuations (especially in the first three lines) overwhelm a procedure that allows the interlinear spacing to change significantly across the page, but only gradually so.

To better assess the respective contributions of these two factors to the failure, the clustering procedure was modified slightly, in order to use not only the nodes retrieved from the segment intersections, but also some pseudo-nodes matching the middle-pixel of every free segment. Even with this much-increased nodes population, the clustering algorithm performed badly (Figure 4.3).
Figure 4.1: Segmented manuscripts
Figure 4.2: Failed segmentation (Ms. MSB 23, with regular nodes): the algorithm does not fare well with fluctuating lines; note the sparse node population.

Although our reduced set is too small to draw definitive conclusions, Table 4.2 shows that the program designed in this thesis either performs very well or fails catastrophically. This behavior is mainly due to the fact that the clustering algorithm, the core of the process, has no nice failure mode. Any large error on an axis estimate (such as an axis crossing two handwritten lines) during the first iteration is usually propagated to the lines to come; the subsequent iterations only make marginal corrections to the resulting clusters. Since segment assignment is strictly determined by the nodes’ labels, no correction can happen
Figure 4.3: Failed segmentation (Ms. MSB 23, with pseudo-nodes): the algorithm does not fare well with fluctuating lines.

As shown by samples 1 to 7, the segmentation procedure described in this thesis meets the expectations for a wide range of unruled manuscripts, provided they are still relatively well-behaved. The case of Ms. MSB 23 shows however that it is not likely to perform well when applied to documents that are out of its intended target.
4.3 Possible improvements

4.3.1 Residual errors

Most errors in the segment assignment phase are not related to the quality of the clustering operation. Therefore, a number of typical mislabeling cases could be solved by refining the heuristics that determine the segment assignment. The mislabeling of abbreviation marks shown in Figure 4.4, for instance, could be corrected with only minimal changes to the assignment rules, taking into account the horizontal shape of the segment’s enclosing box and its position in the core strip. Segment shape and location information could be applied to other mis-assigned patterns, thus helping improve an assignment procedure that already performs satisfactorily.

Noise in the thinned manuscript is also a cause for errors. A single spur on an abbreviation mark, for instance, makes it a 3-segment composite object: the robust assignment rules for free segments do not apply anymore. On faded and degraded manuscripts, performance gains could therefore also be obtained by optimizing the main source of noise in the thinned segments, i.e., the binarization process. This thesis applies only straightforward techniques that leave much room for improvement.

4.3.2 Genericity

Extending the range of acceptable inputs for this segmenting procedure could be made in three ways.
Improving the ability to handle different scripts  This is the easiest change one can bring to the procedure. As mentioned briefly in Section 4.2.3, the node-and-segment approach is by no means restricted to cursive scripts, although the work is made easier by the numerous stroke intersections. Nodes can easily be created from any pixel in any segment, as in a classical pixel sampling procedure, with the added ability to restrict the sampling to specific segment types.

Making the clustering algorithm more resistant to fluctuations  The two assumptions of the clustering algorithm (quasi-periodicity and well-behaved lines) allow for an efficient clustering method relying on least square estimates. The ability to store the line axes as linear functions makes it particularly easy to develop segment assignment heuristics based on the segment position with respect to a given line axis, at little cost. A significant drawback is that the estimates of distinct, fluctuating lines can intersect in a tightly packed manuscript (as in Ms. MSB 23), thus derailing the clustering operation. There is no obvious solution to this difficulty. Using quadratic estimates would only add instability without providing a solution to a rapid variation of the interlinear spacing between to pairs of lines. A more promising approach could use the local vertical projections of the left-hand and right hands part of the page strip under consideration to help determine the next line estimate.

Improving the ability to work with degraded manuscripts  Even if our technique yields good results on faded inks, there is no doubt that a segmentation procedure based on feature-extraction is very dependent on the quality of the preprocessing phase. To be able to handle heavily degraded documents, our segmenting pipe-line should include a much more sophisticated binarization procedure.
4.3.3 Restoring the character strokes from the thinned segments

In spite of the limitations mentioned above, this segmenting procedure is still able to handle a wide range of manuscripts very well. The strong association of the segmentation procedure with the thinning step is a more fundamental drawback. With a 1-way thinning algorithm, there is no easy way to restore the segmented lines to their original shape, thus giving the user the freedom to employ any kind of treatment thereafter. This is a significant constraint on the character segmentation phase that normally follows line segmentation.

Adding the option to regrow the characters after the line segmentation would be a significant improvement to our method. Those connected components whose segments carry the same label are easy to restore to their original shape, simply by filling in the foreground pixels around the segments. Touching components, i.e. components made of segments with different labels (Figure 4.5), would require more work. A distance-based skeletonization theoretically permits the restoration of the characters’ shape after separating the lines. However, distance-based skeletons usually do not have the same morphological properties that make erosion-based skeletons so useful to our purpose: they are not as intuitive a representation of the stroke and the current segment assignment heuristics would not apply. A dilation-based method would be worth exploring. Reversing the thinning, labeled segments can be dilated iteratively, until they reach the original pattern boundaries.

Figure 4.5: A mixed component: obtaining a segmented non-thinned pattern from labeled segments is not a trivial problem.
Such a gradual process would make it easier to control the labeling of pixels in the regions where segments with different labels meet.

4.4 General conclusions and further researches

This thesis describes a new line segmentation method that is optimized for medieval manuscripts. Using a thinned version of the binarized document image, the segmentation algorithm extracts two types of salient features from the handwritten patterns: nodes, whose distribution allows for the detection of line axes; segments, which are labeled according to the nodes they connect. This method obtains very good results on manuscripts that are usually considered hard to segment because of the numerous overlapping and touching lines. By contrast with many existing segmentation algorithms, this method does not rely on user-entered parameters and is not overly sensitive to the quality of the preprocessing treatments. Although more work is required to make it resistant to fluctuating lines, this line separation technique can already handle a large set of medieval documents and provides a useful input to a character segmentation program.

Indeed, late medieval scripts are still a rich source of problems for OCR researchers. In spite of its exploratory character, this thesis intends to provide a starting point for a more complete OCR toolset designed to handle this challenging material.
Bibliography
BIBLIOGRAPHY


aratro crucis applicatus est quasi obtemperans legi predi-centi quod crucem esset pas-surus. Ipse aries dux ouium ecclesie. Item hyrcus pro similitudine carnis peccati nam hyrcus hirsutus et fetidus designat peccatum. Item pro peccato iudeorum et gentilium ymmolatus est quasi vitulus pro iudeis et hircus pro gentilibus. Vel vitulus et hircus dicitur quia pro peccato iustorum et peccatorum ymmolatus est. Dum enim presens vita agitur sine peccato etiam iusti non sunt. Item pro adipiscenda perfectione virtutum est ymmolatus in holocaustum quasi aries. Item pontifex Christus orauit (Ioh 17) Primo pro se: "Clarifica me pater apud temet ipsum claritatem quam habui prius quam mundus esset". Secundo pro domo sua id est pro ecclesia vel pro apostolis domesticis sui (d) "Pater sancte serua eos in nomine tuo quos dedisti mihi". Tertio orauit pro omni cetu fidelium: "Unde non pro hiis tam rogo sed et pro eis qui credituri sunt per verbum eorum in me". Item velum est caro Christi quod tecta fuit di-vinitas intus quod est sanctus sanctorum nosque sumus ex-terius. Unde velum car-nis Christu est medium int-
A.3 Ms HRC leaf M3, f96r: binarized

\[
\text{\ldots}
\]

\[
\text{\ldots}
\]
A.4 Ms HRC leaf M3, f96r: connected components map
A.5 Thinning with BM2+4-pass procedure
APPENDIX B

Test manuscripts
Figure B.2: Ms. HRC 153, University of Texas, Austin
Figure B.4: Codex 1068, Dombibliothek, Cologne
Figure B.5: Codex 1086, Dombibliothek, Cologne
Figure B.6: BANC MS UCB 85, UC Berkeley
Figure B.7: MSB 23, John Hopkins University, Baltimore
APPENDIX C


Zhang and Zuen’s procedure [43], is one of a plethoric family,\(^{18}\) often chosen because of its conciseness and its satisfactory performance on a variety of inputs.

**Algorithm C.10** Parallel thinning algorithm (Zhang-Zuen, 1984 + Lü-Wang, 1986)

1: function \(\text{TTHIN-ZANG}(\text{Img})\) \(\triangleright\) Input is a binarized image
2: repeat
3: \(\text{changed} \leftarrow \text{false}\)
4: \(\text{for pass} \leftarrow 1, 2 \text{ do}\)
5: \(\text{for all foreground pixel } P_1 \text{ in } \text{Img} \text{ do}\)
6: \(C_1 \leftarrow (\text{card}(\text{neighbors of } P_1) \leq 6)\)
7: \(C_2 \leftarrow (\text{card}(\text{neighbors of } P_1) \geq 2)\)
8: \(C_3 \leftarrow (\text{card}(\text{transitions } 0 \to 1 \text{ in sequence } (P_2, \ldots, P_9)) = 1)\)
9: \(\text{if } \text{pass} = 1 \text{ then}\)
10: \(C_4 \leftarrow (P_4 = 0 \text{ or } P_6 = 0 \text{ or } (P_2 = 0 \And P_8 = 0))\)
11: \(\text{else if } \text{pass} = 2 \text{ then}\)
12: \(C_4 \leftarrow (P_2 = 0 \text{ or } P_8 = 0 \text{ or } (P_4 = 0 \And P_6 = 0))\)
13: \(\text{else}\)
14: \(C_4 \leftarrow \text{false}\)
15: \(\text{if } (C_1 \And C_2 \And C_3 \And C_4) \text{ then}\)
16: \(\text{Mark } P_1 \text{ for deletion}\)
17: \(\text{changed} \leftarrow \text{true}\)
18: \(\text{for all foreground pixel } P \text{ in } \text{Img} \text{ do}\)
19: \(\text{if } P \text{ marked for deletion then}\)
20: \(P \leftarrow 0\)
21: \(\text{until } \text{changed} = \text{false}\)

**Overview**  The algorithm iterates over the image (l. 2), deleting 2 layers of boundary pixels in each iteration. In both passes (l. 4), a \(3 \times 3\) neighborhood mask \(N_8()\) is applied to each pixel.

\(^{18}\)More than 100 thinning algorithms have been published.
\[ P_2 = 0 \land P_8 = 0 \]

(a) \[ P_4 = 0 \]

(b) \[ P_6 = 0 \]

(c) \[ P_2 = 0 \]

(d) \[ P_4 = 0 \land P_6 = 0 \]

(e) \[ P_8 = 0 \]

Figure C.1: Zhang’s algorithm: constraints for passes 1 (a-c) and 2 (a-b)

\( F \)-pixel \( P_1 \) and the local connections to its 8 neighbours are examined. If a pixel fulfills the conditions defined for the current pass (l. 15), it is marked for deletion (l. 16), but no modification occurs until all \( F \)-pixels have been examined. Then deletable pixels are suppressed in a single step (l. 20). Iteration ends when no change has been performed in either pass. Although the thinned output is mostly made of 8-connected \( F \)-pixels, Figure 3.6-b shows that the procedure sometimes fails to obtain the thinnest possible segments, preserving unwanted 4-paths in some places (e.g. the tail of the "p").

**Permanent conditions for deletion** In the marking operation, connectedness properties are evaluated in the \( 3 \times 3 \) context \( N_8(P) \). Although disconnected \( F \)-components or \( F \)-components in the mask might be part of the same connected component in a larger window, preserving the topology \( N_8(P) \) is necessary to maintain the image most relevant topological features. Condition \( C_1 \) (l. 6) states that \( P_1 \) is 4-adjacent to at least one \( \bar{F} \)-component, so that its deletion will not result in a cavity in the \( F \)-pattern. Condition \( C_3 \) (l. 8) ensures that \( P \) is adjacent to at most one \( \bar{F} \)-component so that its deletion will not merge two \( \bar{F} \)-components. Condition \( C_2 \) (l. 7) avoids the deletion of a segment end.
Figure C.2: $3 \times 3$ contexts for deletable pixels (pass 1 only - S and E erosion)

Pass-specific conditions for deletion  By condition $C_4$ on line 10 (see Figure C.1-(a-c)), the deletion mark is applied during the 1st pass to

- East and South facing embedded pixels (see C.2)

- all corner pixels: conditions $(P_2 = 0 \land P_8 = 0)$ allow for deletion of NW corners, while conditions $P_4 = 0 \lor P_6 = 0$ permit the deletion of NE, SE, and SW corners (see Figure C.3-(a-f)).

- all ear-pixels: North border pixels, by conditions $(P_2 = 0 \land P_8 = 0)$; East, South, and West border pixels, by conditions $P_4 = 0$ and $P_6 = 0$ (see Figure C.3-(g-j)).
Similarly, by condition $C_4$ on line 12 (see Figure C.1-(d-f)), the deletion mark is applied during the 2nd pass to

- West and North facing embedded pixels (see C.4)
- all corner pixels: conditions $(P_4 = 0 \land P_6 = 0)$ allow for deletion of SE corners, while conditions $P_2 = 0 \lor P_8 = 0$ permit the deletion of NW, NE, and SW corners (see Figure C.3);
- all ear-pixels: South border pixels (by conditions $(P_4 = 0 \land P_6 = 0)$); West, North, East border pixels (by conditions $P_4 = 0$ and $P_6 = 0$).

Therefore, the only directional constraint that guides erosion in each pass applies to horizontal and vertical boundaries: E vertical edges and S horizontal edges are entirely erased after completion of pass 1, including their end corners; W vertical edges and N horizontal edges are entirely erased after completion of pass 2. Both passes erode all ear-pixels and all corner pixels. The algorithm overall performance match these features:

- the morphology of vertical and horizontal stroke patterns, and to a less extent, of curved strokes, is well preserved;
Figure C.5: Zhang and Zuen’s algorithm: failure cases

- since ear-pixels get erased in every pass, the final result is not too sensitive to peripheral noise on the pattern boundaries, which otherwise results in spurs on the thinned characters;

- some specific, diagonally symmetric patterns can vanish entirely (squares), or be reduced to a single pixel (diagonal segments), as illustrated by Figure C.5.

Increasing the minimum number of neighbors for a deletable pixel, from 2 to 3, as suggested by Lü and Wang [46], solves the problem of diagonal segment single-pixel com-
pression. Although the vanishing patterns are specific enough so that they are unlikely to be found in a handwritten document, this topology-breaking behavior is worth some attention.
APPENDIX D

Implementation

All treatments described in this thesis use our own implementation, with two notable exceptions: the Fast Fourier Transform and the local linear regression (Chapter 2), that use the standard R routines. Most exploratory work for this thesis used C++. The CImg library[57], an image processing library, provided routine input/output functions and a number of convenient iterators. A prototype for the segmentation chain was developed as an extension of this library. It comprises the most important treatments described in these pages, including the thinning and clustering operations. This implementation is reasonably efficient, while providing some ways to visualize the intermediate data: node clusters can be displayed, for instance, as well as other salient features (segments, node vicinity maps, connected component enclosing boxes, line axes etc.). However, options are limited and are a poor substitute for a dedicated plotting program, such as R.

Initially developed as an open-source implementation of S,19 R [58] is optimized for numerical treatments, with powerful plotting capabilities. Thanks to a rich library of user-contributed packages, R can also be used as a general-purpose scripting language. Re-implementing the treatment chain in R made it much more convenient to provide the many plots that serve as a visual counterpart of the treatment descriptions contained in this report, such as projections, details of the thinning operations, and annotated representations of the clustering steps.

19The data analysis language S was created by John Chambers in 1998 at the Bell Laboratories. Its early commercial implementation, S-PLUS, has been largely superseded by R in most research institutions.
D.1 A simplified view

The whole segmentation chain is a collection of R scripts put together in a Makefile. Figure D.1 provides a simplified view of the processing line. The different states of the input document image are represented with red boxes: [.jpg] is the input image; [8cols.png] is its 8-color version; [binarized.png] is the binary document; [thin.png] is the thinned manuscript; [separated_lines.png] contains the segmented lines.

The elliptical boxes stand for the R scripts. Each of them performs a single step of the treatment, generating some intermediate data to be used by the next script. The actual segmentation process occurs in two programs:

- <clusters.R> reads the array of node coordinates ([.ctr]) and the result of the Fourier analysis of the profile ([.fft]) to classify the nodes into clusters, storing the result in a table mapping each node to a cluster ([.n2c]) and the array of linear functions [.f1].

- <segment_assignment.R> uses these data as well as a file [.sgt] describing the segments (as lists of pixels) in order to classify the segments among the clusters, applying the heuristics described in Section 3.3.2.2 of this report. The data structure generated by this operation is a table mapping each segment to a cluster ([.s2c]), that is used in a subsequent step by <line_separation.R> to color the segments accordingly.

The remaining scripts perform the preprocessing steps needed to obtain the thinned image and extract its salient features: <otsu.R> transforms the input into an 8-color file; <mono.R> generates the binary image; <thin.R> produces the thinned patterns. Nodes and segments are retrieved by <nodes.R> and <segments_retrieval.R>, respectively.
Figure D.1: The segmentation pipe-line: a simplified view
D.2 The detailed view

A number of auxiliary treatments are actually involved in most phases of the segmentation chain. The picture of the segmentation chain shown in Figure D.2 includes those secondary steps. The clustering algorithm requires an initial estimate of the text block boundaries [.ths] computed by the program <block_boundaries.R>. The connected component analysis performed in <cc.R> yields a connected component map ([.cc]) and a table containing the coordinates of the corresponding bounding boxes ([.box]). The segment assignment routine uses these data to estimate the position of the patterns with respect to the line axes. The program <free_segments.R> reads the list of internal and external segments [.sgt] to compute (by subtraction) an image containing only the free segments, which are then serialized (as lists of pixels) in the file [.fsgt] and finally attributed a cluster label through specific rules: the result is stored in the table [.fs2c] that maps every segment to a cluster.
Figure D.2: The segmentation pipe-line: complete view