Vector-attribute character recognition

Steven Bouma, Werner Buck, Ruurd Moelker

September 2, 2011

Bachelor thesis in Computing Science
University of Groningen
Faculty of Mathematics and Natural Sciences
Computing Science
September 2011

Authors:

- Steven Bouma, Bachelor Computing Science, s.bouma@rug.nl
- Werner Buck, Bachelor Computing Science, w.e.s.m.buck@student.rug.nl
- Ruurd Moelker, Bachelor Computing Science, r.r.moelker@student.rug.nl

Supervisor:

- Michael H. F. Wilkinson
- M.H.F.Wilkinson@rug.nl
- http://www.cs.rug.nl/~michael/

Secondary supervisor:

- Arnold Meijster
- A.Meijster@rug.nl

Location:
University of Groningen,
Groningen, The Netherlands
Department:
Faculty of Mathematics and Natural Sciences,
Computing Science

Abstract

Traditional optical character recognition (OCR) uses thresholding and predefined characters to recognize text. This study makes the assumption that thresholding can cause loss of potentially valuable information. A more versatile approach to character recognition with minimal thresholding is offered which relies on a max-tree and image moments. The max-tree is a tree representation of the image where each level represents a grey-level and each node contains attributes. The image moments are scale and translational invariant, this allows recognition of a wide range of characters instances. Finally supervised learning is used to classify the characters, and line and word segmentation is used to obtain a full textual reconstruction of an input image.

Keywords: OCR, max-tree, image moments, vector-attribute filters, supervised learning

Contents

1 Introduction **6**
- 1.1 Optical Character Recognition . . . 6
- 1.2 Research question . . . 8
- 1.3 Program . . . 9
- 1.4 Previous work . . . 10

2 Theory Fundamentals **11**
- 2.1 Max-Tree . . . 11
  - 2.1.1 Max-Nodes and attributes . . . 11
  - 2.1.2 Max-Tree vs. Min-Tree . . . 11
  - 2.1.3 Construction . . . 11
  - 2.1.4 Basic example . . . 14
  - 2.1.5 Construction complexity . . . 15
- 2.2 Image moments . . . 16
  - 2.2.1 Centroids . . . 17
  - 2.2.2 Translational invariance . . . 17
  - 2.2.3 Scale invariance . . . 18
  - 2.2.4 Rotational invariance . . . 18
  - 2.2.5 Image moments overview . . . 19
- 2.3 Vector similarity . . . 19
  - 2.3.1 Distance measures . . . 19
- 2.4 Max-Tree filtering . . . 21

3 Classification **22**
- 3.1 Character set . . . 22
- 3.2 Selection Methods . . . 22
  - 3.2.1 Flood select . . . 23
- 3.3 Selection method conclusion . . . 26

4 Model construction **27**
- 4.1 Class/character . . . 27
- 4.2 Prototype . . . 28
- 4.3 Multiple node prototype . . . 30
  - 4.3.1 Broken characters . . . 30
  - 4.3.2 Multiple segments . . . 31
  - 4.3.3 Developed Solution, multi-select . . . 31

5 Recognition **32**
- 5.1 Single node recognition . . . 32
  - 5.1.1 K Nearest Neighbours . . . 33
  - 5.1.2 Other learning algorithms . . . 33
  - 5.1.3 Prototype grey-levels within range. . . . 33
- 5.2 Multiple node recognition . . . 34
  - 5.2.1 Different approach . . . 34

5.2.2 Desired solution 34
5.3 Collapse 35
5.4 Collapse Algorithm 36
5.5 Best node selection 42
5.5.1 Best matching (lowest distance) 42
5.5.2 Longest chain 43
5.5.3 Summary 45
5.6 Complexity 45
5.7 Recognized character to text 45

6 Findings **46**
6.1 Clear printed text 46
6.1.1 Trained page 46
6.1.2 New page 48
6.2 Historical printed text 49
6.3 Method Conclusion 52

7 Future work **52**

1 Introduction

Researchers have been looking for a convenient way to annotate historical documents for some time. The documents are often in bad shape, suffering from backbleeding, smears and general degradation. Because of the continually degrading quality there is a growing need to preserve the contents of these old documents. It takes manual inspection to translate these frail documents to machine recognized text. There is hope however. In 2010 Google estimated that there are about 130 million unique books in the world (129,864,880 to be exact), and that it intends to digtilize all of them by the end of the decade [15][9]. This shows that there is demand for a fast and reliable method of converting printed text to digitized text. The process of converting images (scans) of printed text to digitized text is called Optical Character Recognition (OCR). First the classic approach is discussed then the studied new method is presented.

1.1 Optical Character Recognition

In order to meet the goal of digitizing all unique books, Google uses multiple forms of OCR. OCR is the mechanical or electronic translation of scanned images of handwritten, typewritten or printed text into machine-readable text.

OCR is typically achieved through a number of steps.

1. Document input
2. Image preprocessing
3. Document & Layout Analysis
4. Recognition
5. Verification & User interaction
6. Export/Document Output

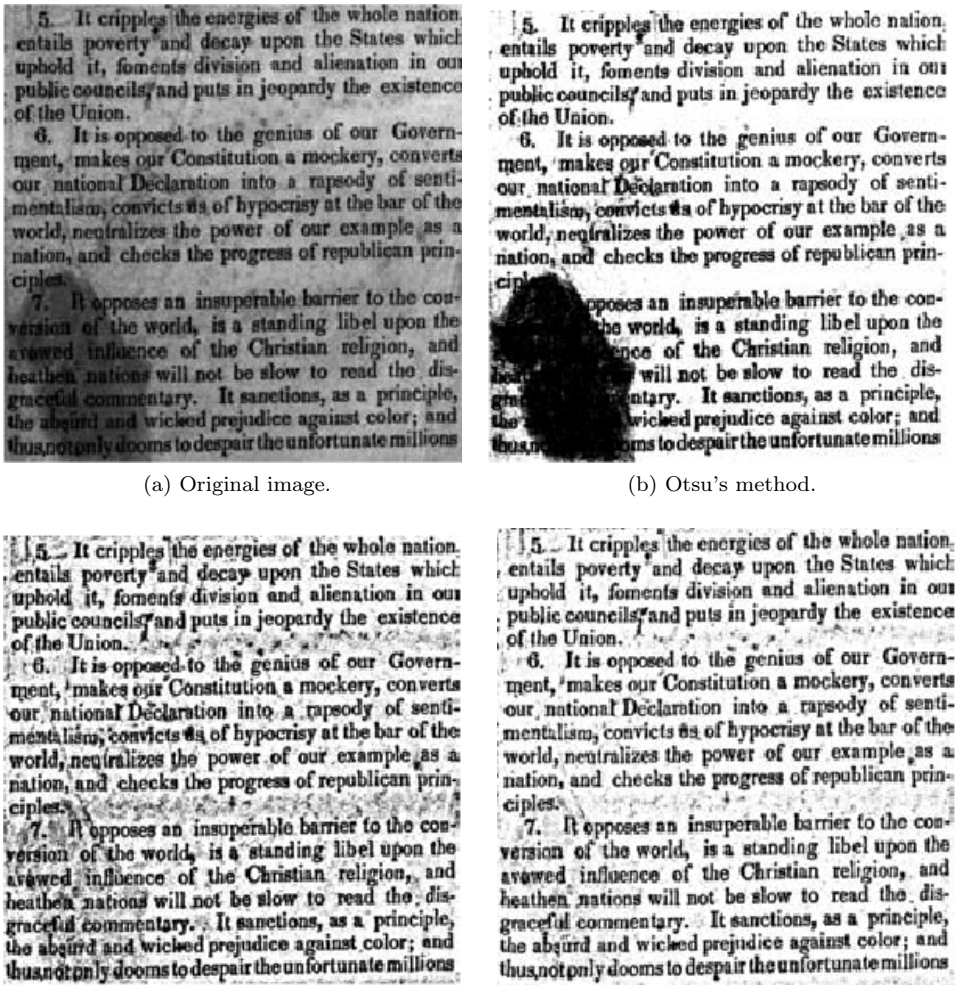
The steps "Image Preprocessing" and "Recognition" are the most interesting steps in the context of this thesis.

Image preprocessing often starts with a skew correction to ensure characters are aligned horizontally. After this correction the document can be cleaned and filtered to remove noise and other forms of degradation to prepare it for recognition. When starting recognition, characters from a template-set, usually black and white are correlated against a particular character. When the character matches a character with enough confidence it is recognized.

In order for the comparison to work fast, most OCR algorithms only accept black and white input images in the recognition phase. Transforming an greyscale input with each pixel having a value range of 0 to 255 to a resulting image with a value range of 0 and 1 (respectively black and white) is called binarization.

Binarization can be a result of various thresholding techniques. Figure 1 shows binarization of a typed manuscript using common thresholding techniques

which clean up document degredation and noise. By thresholding images, noise and other forms of degradation can be reduced. The reduction in noise and overal degredation helps to get a higher correlation confidence in the recognition step.



(a) Original image.

(b) Otsu's method.

(c) Niblack's method.

(d) Souvola's method.

Figure 1: Various thresholding techniques that deal with image defects differently.

Loss of information

For an input image, the higher the resolution and scan quality the better the resulting confidence of correlation in OCR. The more information the various filters and thresholder techniques have to work with the better the outcome.

Thresholding works by making decisions about what is important, and discards information that it thinks is not useful. The information left after thresholding is then used by the OCR's recognition step.

Whether the information that is discarded is important depends on the thresholding technique used. And as can be observed from figure 1the result can be quite different. Especially observe how Otsu's method in figure1b actually makes text under the stain unreadable.

Despite this potential loss of information conventional OCR methods average to 95 percent accuracy and with help of a lexicon it can average to 98 percent. Which means that an average page of 400 words can have as many as 20 mistakes. [7]

1.2 Research question

The hypothesis of this thesis is that an alternative method for optical character recognition is possible based on recognition with image moments using a Max-Tree [13]. The proposed method applies to greyscale images and should keep thresholding to a minimum to avoid loss of information. In addition, the method should allow a user to define a character set and save it as a model, and should not require any predefined knowledge of characters. In relation to the OCR pipeline mentioned earlier, this means step 2 pre-processing is replaced by the learning of "ground-truth" characters. Because some form of skewness correction has already been done on the documents, step 3 of the pipeline is skipped.

This thesis describes and demonstrates this different method of OCR by use of Max-Trees and image moments. This study attempts to answer the question:

> To what extent do image moments provide good character recognition on Max-Tree represented greyscale printed documents?

Max-Tree

The Max-Tree is researched as the possible datastructure for effectivly storing, manipulating and searching grey-value images in the context of OCR [13]. Other studies have shown that the Max-Tree is an effective tool in the area of image recognition.

The Max-Tree is a hierarchical decomposition of the input image. The hierarchical decomposition matches the shape of a tree where each level represents a grey-value level. Details regarding Max-Tree features and implementation are discussed in section 2.1.

Image Moments

Image moments are mathematically calculated attributes or properties of image regions that can be used to identify characters in grayscale images. Many transformations to image moments have been proposed to give these new moments certain properties. A wide range of research is available as image moments have been succesfully applied in several other areas of image recognition. Details regarding image-moment calculation and comparison are explained in section 2.2.

Scope

The method described in this thesis is aimed at machine printed historical documents, as this type of document is not easily recognized using the traditional methods for OCR. Even though not studied, the techniques mentioned may also provide positive results in other areas of image recognition for example, hand-writing recognition.

This thesis researches a method of OCR that uses the original grey-value input image as it's basis and is accompanied by a demonstration program of OCR. Most of the time and effort has gone into the demonstration program which has been built up from scratch in exception of a few utility classes.

Format

This bachelor thesis is written as a report of our findings and experiences over the six month course of development of the program. Thus this thesis also provides a description of the demonstration program that was constructed by the authors of this thesis.

1.3 Program

To test the methods and demonstrate the results a annotation tool is created. This annotation tool program associated with this thesis, contains a new method for annotation of historical documents, based on scale-invariant attribute filters. In this program the user can train the program by giving a number of "ground-truth" letters in a document, after which the system automatically gives the most likely candidates for similar letters.

The demonstration program has largely been constructed from scratch and many man-hours have gone into the creation of this program.

1.4 Previous work

Historical document recognition

T. van Laarhoven has shown in his master thesis [12] that by using a method of OCR that uses correlation, learning algorithms, contrast stretching combined with user feedback can give considerable good recognition on historical documents.

The author notes that the algorithm performs well, yet has difficulties, among others, with symbol distinction because of variable white space size and body selection because of variable margins.

Although very different in approach, this study is derived from his work. A different recognition method is used and compared with T. van Laarhoven methods to see if this method could give a better recognition. The method used in this thesis does not use correlation like in the thesis of T. van Laarhoven's. Instead the method uses more information (multiple grey-levels) to more accurately recognize the characters.

Moment assisted character recogition

A study by El Khaly and Maher [2] has already studied the moment assisted pattern recognition on Arabic printed text. The study has achieved good recognition on systems that obviously bleek in comparison to their modern counter parts. An important observation from this study is that using normalized central moments, small differences in line thickness could yield fairly great changes in the image moment and resulting recognition. The study also focuses on recognizing printed characters yet add some new principals.

This thesis study compared to the study of El Khaly and Maher have numerous differences. This study consists of recognizing entire pages full of characters also to a lesser extend with newlines and white spaces recognition. Next pur approach focusses on western style text with letters instead of symbols. This has the consequence that there is no need for symbol separation as is the case for arabic characters. Finally the aim of our method is to be capable of recognizing any set of printed characters, or separated symbols for that matter, which is done by an intensive period of learning from user feedback.

2 Theory Fundamentals

The main building blocks used in the new OCR method are explained in detail in this chapter.

2.1 Max-Tree

The first step in OCR is image decomposition. A grey-level scan needs to be stored in a way that it can be filtered, changed and read out quickly.

The Max-Tree as proposed by Salembier et al [13] provides a good data structure for storing a image decomposition and holding additional attribute collections per node.

2.1.1 Max-Nodes and attributes

As with any tree structure, the Max-Tree consists of nodes (called Max-Nodes in this case). During construction, the Max-Tree is built up from the root up recursively for each grey-level.

Each level has nodes. A node contains information on the part of the input image that the node represents. The information that the node contains are also called attributes. Node-attributes can contain basic properties such as grey-level, pixel-area and parent-child relations, as well as more specialized vector attributes that can be used for vector-attribute filtering [16].

2.1.2 Max-Tree vs. Min-Tree

A Max-Tree is oriented towards the maxima of an image (the leaves of the tree are the lighter components in the image and the root being the darkest), hence the name Max-Tree. A Max-Tree was used by Salembier et al. to decompose photographs which have bright structures on a dark background. In this case the Max-Tree is intended to be used for OCR, where (it is fairly safe to assume) a paper of lighter background with darker characters printed on it is used. The Max-Tree is therefore inverted in this case - with level 255 (white) being the root node, and each level h representing the thresholded image for grayvalues $\geq h$.

As a result of the inversion the dark parts, (which likely are characters) are located near or in the leaves of the tree. This inverted Max-Tree is referred to as a Min-Tree , and called as such as it is oriented towards the minima (darker parts) of an image. [13]

The Max-Tree is in every other aspect the same as a Min-Tree therefore in the next chapters this thesis will refer to the tree used as a Max-Tree while it is in reality a Min-Tree and more generally both are an component-tree [10].

2.1.3 Construction

The Max-Tree is constructed utilizing the flooding-algorithm as described [13] by Salembier et al. The idea consists in recursively thresholding the image

at all possible grey-levels starting from the lowest (or highest for a Min-Tree) pixel-intensity of the image.

The method shown in Algorithm 1 does not yet include attributes. It is merely an algorithm for the construction of the tree and the node-relations. In this case and in the method of use in this thesis each node is a flat zone with the same intensity. This however this is not necessarily a requirement for the Max-Tree, as Salembier et al also mentioned that each zone can represent more grey-levels. For example one might use a pixel-intensity $\triangle > 0$, resulting in zones consisting of multiple intensities, these are called K-flat zones [13].

A modified version of the flooding algorithm is shown in Algorithm 2 and includes the creation and merging of node attributes. In this case the actual pixel coordinate (x,y) becomes important as it is needed for calculating the node's attribute (details are discussed in the section on image moments). In addition parent area and parent attributes are carried along in the recursion, which allows for the merging of the child data with the parent data. The following pseudo-code algorithm constructs a Min-Tree and allows for computation of raw image moment data.

Algorithm 1 Flooding without attributes.

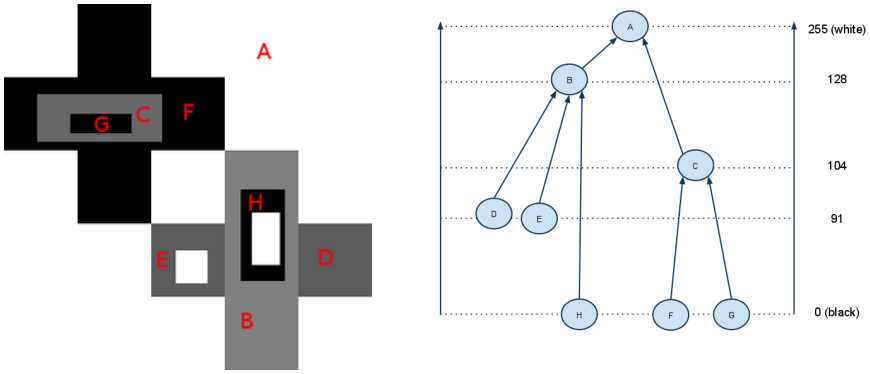
```
flood (h)
  while not hqueue-empty(h)
  p = hqueue-first(h)
  status(p) = number-nodes(h)      /* Process p */

  for every neighbor q of p        /* 4 or 8 connectivity */
    if status(q) == NOT_ANALYZED
      hqueue-add(image[q], q) /* where image[x] is the graylevel at (one-
          dimensional) index x */
      status(q) = IN_THE_QUEUE
      node-at-level(image[p]) = true
    if (image[q] >image[p])        /* We have found a child at level q */
      m = image[q]
    repeat
      m = flood(m)               /* Flood the child */
    until m = h

  number-nodes(h) += 1

    m = h - 1
    while m >= 0 and node-at-level(m) = false
      m --           /* look for the father */

    if m >= 0
      i = number-nodes(h) - 1
      j = number-nodes(m)
      Cih.parent = node(Cjm)
    else
      this is the root /* no parent */

    node-at-level(h) = false
  return m
```

Algorithm 2 Flooding with attributes.

```
flood (h, ref thisarea, ref thisattr)

  area = thisarea

  while not hqueue-empty(level)
  {
    area++;
    p = hqueue-first(level)
    x = p % width;
    y = p / width;

    if exists(attribute)
      attribute.add(x, y, level)
    else
      create(attribute)

    if exists(thisattr)
      attribute.merge(thisattr)

  for every neighbor q of p      /* 4 or 8 connectivity */
    if status(q) == NOT_ANALYZED
    hqueue-add(image[q], q)
    status(q) = IN_THE_QUEUE
    node-at-level(image[p]) = true
    if (image[q] >image[p])    /* We have found a child at level q */
      m = image[q]
    repeat
      m = flood(m)        /* Flood the child */
    until m  = h

  number-nodes(h) += 1

  m = h - 1
  while m >= 0 and node-at-level(m) = false
    m --              /* look for the father */

  if m >= 0
    i = number-nodes(h) - 1
    j = number-nodes(m)
    Cih.parent = node(Cjm)
  else
    (*skip*) this is the root /* no parent */

  area += childarea
  attribute.merge(childAttribute)
  node.assign(attribute, level, area)
  valueof(thisarea) = area;
  valueof(thisattr) = attribute;
  node-at-level(h) = false

  return m
```

2.1.4 Basic example

To illustrate the structure of a Min-Tree, the construction results for the following basic example are given below.



(a) A basic Min-Tree example of an 300x300 pixel image.

(b) Min-Tree for figure 2a.

Figure 2: A simple example of Min-Tree construction.

Node Label	Node Index	Level	Cumulative area	parent
A	0	255 (white)	90000 (300x300)	-
B	55691	128	16309	0 (A)
C	63791	104	18000	0 (A)
D	66969	91	3600	55691 (B)
E	66970	91	2898	55691 (B)
F	73467	0 (black)	14022	63791 (C)
G	73468	0 (black)	800	63791 (C)
H	73469	0 (black)	1711	55691 (B)

Table 1: Figure 2 Min-Tree table form.

Explanation The original image is composed of 8 flat zones and 5 different grey-value's. In the first step of construction the threshold h is fixed to the gray level value 255, the image is binarized and all pixels at level h are assigned to the root node of the tree (A). The pixels lower in intensity than level h form two connected components that are temporarily assigned to two nodes: {C,G,F} and {B, D, E, H}.

This creates the first part of the tree namely A being the root and parent of two nodes {C,G,F} and {B,D,E,H}.

After this, the threshold h is decreased by 1, however there are no flat zones with level h = 254 so h is again incremented by 1.

Finally when h is level 128, each node found at that level is processed as the original image. This means that from the node {B,D,E,H} everything lower than level 128 is pushed into a temporary node as a child of B.

At this point the nodes are {A} root, with children {B} and {C,G,F} and {B} has a child node {D,E,H}.

Again h is decremented until level 104 where the same steps are done as before.

This process is repeated until the lowest intensity zero has been processed. The resulting tree then looks like figure 2b.

2.1.5 Construction complexity

This is a sequential algorithm, however several other implementations have been proposed each with different complexities [6]. These include a concurrent approach [18], which allows parallel Max-Tree construction by merging subtrees into a final result.

Since performance related issues are outside the scope of this thesis, the basic sequential version is used. However it is to be noted that performance may be increased by using alternate approaches for construction, filtering and other Max-Tree related operations.

The worst-case complexity of this particular algorithm is $O(N * (C + G))$, with N being the number of pixels in the image, C the connectivity (4 or 8) and G the number of gray-levels (256 in this case) [18]. This worst-case scenario applies to images where the number of nodes is maximal (e.g. every pixel is a separate node), so it is expected that performance will be better in practise.

2.2 Image moments

In a general (mathematical) context a moment is, loosely speaking, a quantitative measure of the shape from a set of points. Moments when applied to digital images are referred to as *image moments.* During decomposition of an image every subset pixels of that image can described by means of image moments. An image moment is a weighted average (moment) of the pixel values in the image (or a subset of the image).

Image moments are stored in the Max-Tree as vector attributes. An image moment represents a set of pixels in each node of the Max-Tree as a numerical vector. By calculating the distance between the attribute-vectors of nodes, the similarity between the two nodes can be calculated.

Raw image moments can be computed in Max-Trees by making use of the commutative property. Each parent of a node holds the summed combination of his own image moments as well as the children's image moments.

Following this line of reasoning, the root node of a tree contains the image moments of an entire image.

Raw image moments

For a 2-D continuous function f(x,y) the raw image moment of order $(p + q)$ is defined as.

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q \, f(x, y) \, dx \, dy$$

with $p, q \in \{0, 1, 2, 3, ...\}$

This however does not apply to the greyscale images that the proposed method works with (as they are not continuous). In the discrete case of greyscale images with pixel intensities img(x,y) raw image moments are defined as:

$$M_{ij} = \sum_{x} \sum_{y} x^i y^j \, img(x, y)$$

Raw moments are also known as *Geometric Moments.*

These raw moments or geometric moments can be transformed into other, more specific moments each having different properties. It is possible to not incorporate the actual greyvalue in which case f(x,y) is simply considered 1 or 0. If this is done the image moment represents the shape of an image (mask), rather than the image itself. These Image moments are computed per node, and each node has at most one greylevel.

2.2.1 Centroids

One of the basic properties of raw image moments is the ability to calculate the centroid. The centroid is the geometric center of a shape, or more formal the intersection of all straight lines that divide the shape into two parts of equal moment about the line.

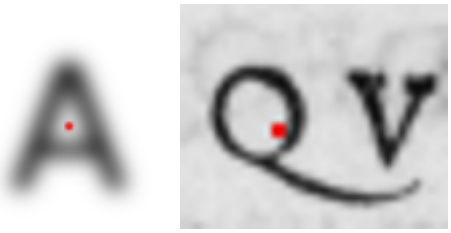


Figure 3: Centroid of characters 'A' and 'Q'.

Based on image moments the centroid is defined as:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} M_{10}/M_{00} \\ M_{01}/M_{00} \end{pmatrix}$$

Image centroids define the position of an image sub-part within an image. Centroids are, amongst other things, used to make image moments invariant to translation.

2.2.2 Translational invariance

One basic desired property of image moments would be *translation invariance.* A raw image moment that was computed on a specific subset of an image will *not* be similar for another subset with the same shape but a different location. This because of the simpel fact that the actual pixel coordinate is incorporated in the raw moment. So for example: a raw image-moment computed for an image-part containing a character 'E' will not be similar for an identical 'E' located elsewhere in the image. Raw image moments are sensitive to translation (movement of all coordinates in some direction), rendering them less suitable for general recognition.

Translation invariant image moments or otherwise known as *central moments,* on the other hand allow for recognition of image-sub-parts and shapes regardless of their location within the input image. This is a property that is generally useful in an image recognition context, especially when dealing with characters. Translation invariance is achieved by subtracting the centroid coordinate from the actual pixel coordinates. For discrete cases this results in the following formula:

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q \, f(x, y)$$

Where $\bar{x}$ and $\bar{y}$ are the centroid x- and y counterparts. This formula is very similar to the one for geometric moments, with the centroid substraction being the only exception.

2.2.3 Scale invariance

Another property useful for recognition using image moments is the invariance to scaling. Scale invariant image moments are consistent between scaled instances of an image or shape. This property is very desirable in case of character recognition, since printed text often contains characters from the same font, but with a different font-size. Scale invariant moments are constructed by normalizing the translational invariant central moments. This is done by dividing all elements $\mu_{pq}$from the central moments by the scaled image area μ_{00}, resulting in:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\frac{(p+q)}{2}+1}}$$

As a result the original central moments are normalized and the image moments are both translation - and scale invariant. These image moments are called *normalized central moments*.

2.2.4 Rotational invariance

None of the image moments discussed so far will suffice for recognition of a skewed instance of a character. Skewing may occur deliberately in documents as *italics*, but also may occur due to image noise and corrections performed during document digitalization. General rotation invariance is a less obvious required property in case of character recognition since characters have a fixed orientation, and the orientation is of importance to the meaning of a symbol. For example a character 'V' may resemble a vertically flipped 'A', and because pixel distributions for both characters are similar, the rotation invariant image moments are likely to be similar as well. However in this thesis three types of rotation invariant image moments are researched as a tool to deal with character skewing, these are: *moment invariants* as described in [8], *complex moment invariants* as described in [4] and Flusser and Suk *affine moment invariants* as desribed in [5].

2.2.5 Image moments overview

In table 2 an overview of different image moments and their characteristics discussed in previous sections is given.

Invariance	Translation	Scale	Rotation	Affine transformation
Geometric Moments	✗	✗	✗	✗
Central Moments	✔	✗	✗	✗
Normalized Central Moments	✔	✔	✗	✗
Hu moment invariants	✔	✔	✔	✗
Flusser & Suk complex moment invariants	✔	✔	✔	✗
Flusser & Suk affine moment invariants	✔	✔	✔	✔

Table 2: Characteristics different image moments.

2.3 Vector similarity

Once computed, the image moments are stored as vectors in the nodes of the Max-Tree. For computing similarity between two vectors a wide range of distance measures can be used. As this is always a pairwise operation for each element of both vectors, vector length is an important factor of computational efficiency. By convention epsilon (ϵ) is used as the allowable distance between vectors that are considered similar.

Image moments of higher orders provide more precision (in theory), but also use larger vectors and thus require more computation.

2.3.1 Distance measures

The following distance metrics are implemented in the program and are mentioned briefly.

Euclidian distance The euclidian distance formula is based on the pythagorean theorem $a^2+b^2=c^2$. For two vectors $\overrightarrow{a}$ and $\overrightarrow{b}$ the euclidian distance is defined as:

$$dist(\overrightarrow{a}, \overrightarrow{b}) = dist(\overrightarrow{b}, \overrightarrow{a}) = \sqrt{\sum_{i=1}^{N}(a_i - b_i)^2}$$

Normalized Euclidian distance

$$dist(\overrightarrow{a}, \overrightarrow{b}) = dist(\overrightarrow{b}, \overrightarrow{a}) = \sqrt{\sum_{i=1}^{N}\frac{(a_i - b_i)^2}{\sigma_{i^2}}}$$

where σ_i is the standard deviation of the i^{th}

Manhattan Distance / L1 distance

$$dist(\overrightarrow{a}, \overrightarrow{b}) = dist(\overrightarrow{b}, \overrightarrow{a}) = \sum_{i=1}^{N} |a_i - b_i|$$

The benefit of this method over the previous two, is that it is computationally less intensive as there is no squaring or squared root in the formula.

Equal element weighted Euclidian distance From the observation that image moments, discussed in section 2.2, value from small decimal numbers up to extremely large numbers, this distance measure is constructed to still weigh all these moments as equal when comparing. This is done normalizing element-wise. Both elements are stretched or shrunk by 1/largest element. After which the resulting distance is normalized. Additionaly, care must be taken to avoid division by zero, not shown here to improve readability.

$$dist(\overrightarrow{a}, \overrightarrow{b}) = dist(\overrightarrow{b}, \overrightarrow{a}) = \frac{1}{N}\sqrt{\sum_{i=1}^{N}((a_i - b_i) * (1/max(|a_i|, |b_i|))}$$

2.4 Max-Tree filtering

Once the image is decomposed, and every node of the Max-Tree contains attribute-vectors (image moments), these attributes along with other attributes as level or area can be used to discard or keep certain nodes.

A filter can be made based on the similarity criterion that uses a distance measure, like those described in section 2.3.1. This form of filtering allows the substraction or isolation of specific nodes.

Based on the shape in the following figure the example geometric moment was computed. (the order being $p + q = 4$)

119	0	3183.85	-4544.29	0	-845.966	6867.29	3124.44	2275.17	-5085.6

In this case *central image moments* are computed for the reference vector and all Max-Tree nodes. By applying a isolating filter and comparing the attribute vectors from the nodes with the reference, the decision can be made with help of epsilon whether to keep or disregard a certain node. The disregarded nodes are then assigned their parent's grey-level effectively merging nodes that are filtered out with their parent.

Figure 4: Binary shape that defines the character "AE"

(a) Input image

(b) Direct filter result image

Figure 5: Direct filter result: the character 'AE' is isolated.

3 Classification

Recognition is done using a set of reference characters. At first, none of these references or ground truths exist yet. It is up to the user to define what symbol matches what character. The creation of these references consist of selecting the image area which forms the character and storing the result as a prototype.

3.1 Character set

Before discussing classification a character must be defined. A character consists of one or multiple symbols. A symbol means a part of a character such as the dot in the "i". In most cases a letter consists of a single symbol; generally only a small subset of a language characters consists of multiple parts. Examples of multiple part characters in the English language include the "**i**" and "**j**". Also many special characters consist of multiple components such as "!", "?", "%".

Other modern languages have diacritic characters such as the "ü" , "á", "ĉ". Many different diaeresis characters exist e.g. "ˆ", "˝" and "¸". These additions to a character can be seen either as an extension of the alphabet in use or as a special variant of the standard alphabet. The latter would mean recognizing the individual elements and combining them to a new class which would greatly increase the problem complexity. Therefore these diacritic characters are seen as completely different characters from their plain equivalents.

In the context of historical documents it can be the case that single symbol characters are split due to paper deterioration, scan noise or stains. Also the use of grapheme characters such as "Æ" or "æ" is more prevalent in historical documents. While these characters are always printed as a single symbol, caution must be taken during recognition to not simply recognize a section of the symbol but rather the the symbol as a whole.

The problem of classification of diacritic or broken characters is explained further in section 4.3.

3.2 Selection Methods

Before a character can be classified, the user should point out which parts of the image consist of a character. A character in an image is formed by a darker character region on lighter background region. With respect to max-tree image decomposition this means the background is region is close to the root of the tree and the dark character region further away from the root.

As discussed, a character is formed from one or more symbols. Selection is explained for each symbol instead for multiple symbols. For a character consisting of multiple components the selection process is simply repeated for each symbol.

Since every pixel contributes to the image moments that define a character, it is important that the selection method is precise. However process of selection may become tedious, and therefore must also be user-friendly to some extent.

3.2.1 Flood select

The first method for selecting a symbol is a flood fill algorithm. It is centered around the idea that selecting by simply clicking on a character takes only little effort for the user and is thus most desirable. Flooding is started from a pixel and continues as long as a certain condition is true. During flooding every node traversed is marked as selected. For the stop criteria different conditions have been tested. These stop criteria are based on a large difference between nodes of either pixel intensity, node area, centroid position or moment value.

Intensity boundary

The simplest flooding condition is based on pre-set pixel intensity boundaries by the user. This method is very flexible, however compared to other techniques this requires additional user input. The advantage is that once proper boundaries are set for a single selection, little to no changes are needed for the following selections.

This method also does not require any knowledge of preceding pixels and of such can be implemented iteratively. An iterative implementation gives not only a performance boost but can also handle huge selections which can easily cause an overflow in a recursive implementation.

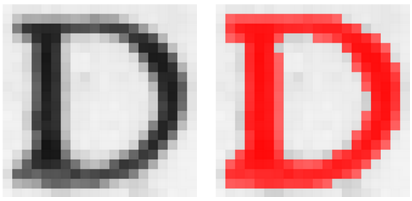


Figure 6: Selection of character “D” from a selected pixel within the character and intensity boundaries [0 - 190].

A requirement for this type of selection is that the character is actually connected by pixels within the given range. For characters such as an ‘i’, that consist out of multiple parts, a single flood-selection of any form is not sufficient. Also a problem may occur when an image is deteriorated or contains high levels of noise. In such cases characters may be split in multiple parts while actually being a whole originally. As displayed in figure 6, the selection of the (latin) character ’s’ fails, because the top part is not connected to the rest due to image distortion. Boundary adjustment does not lead to selection of the entire character, but instead results in overflow and selects neighbour characters and background noise.

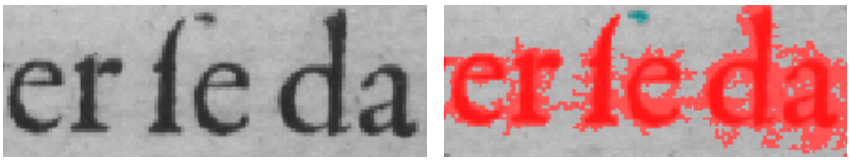


Figure 7: Example of single flood-select failure. The latin character for 's' is never fully covered by the floodselection.

A practical solution for this particular problem is to allow multiple selections. A user can define multiple parts that form the character, and each part can be selected with the flooding algorithm.

Intensity difference

From observation one can see that the pixel intensity of edge pixels decrease more rapidly than center pixels. Using a criteria which allows for a small difference in intensity during flood the character boundary can be found automatically. In many cases this leads to almost full selection of the character. Often however, the pixels outside the character are reached by the flooding algorithm and as a result almost the entire image is marked as selected as seen in the image below.

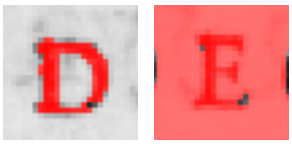


Figure 8: Selection characters "D" and "E" with intensity difference smaller than 50.

Area difference

The area of a node is larger near the boundary of a character with the background. This can be used to flood until the area difference between nodes of subsequent pixels becomes too large. This method fails however, because area differences inside a character are very similar to those outside the character. From the image below one can view how either to little or to much of the image gets selected. Only very occasionally selection with this method succeeds, yet not enough to be of use in practice.

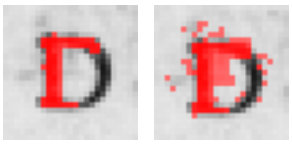


Figure 9: Selection of character "D" from pixel within character using area difference criterion. Area difference threshold for left and right image are respectively 40 and 50 pixels difference.

Centroid shift

Similar to the area based criteria flooding method. The centroid can be used to determine large increases in value. This method however is more sensitive to areas further removed from the current center. Background nodes in general are more elongated and therefore give a greater shift of the image centroid as opposed to nodes inside the character. Therefore it quickly stops the flooding algorithm when the first parts of the background are traversed. Yet this method still suffers heavily from over selecting as seen in the image below. Also background elements which blend smoothly and are a prolongation of the letter are sometimes accepted when they should not, as depicted in the character "E" below.

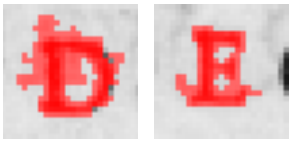


Figure 10: From left to right, selection of characters "E" and "D" from point in character for an accepted centroid shift of 5 pixels.

Moment shift

Since image moments provide a good description of an image, differences in image moments between nodes of subsequent pixels have been used to determine selection bounds dynamically. From all attributes tested the centralized image moments give the best boundary description. As seen in the image below, the "E" and many characters with it are selected well, albeit with some background noise attached. Other characters such instances of the **"D"** are not at all correctly selected no matter what threshold is selected.

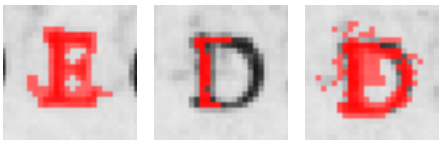


Figure 11: From left to right flood select for characters “E”, “D”, “D” using centralized moment difference smaller than respectively 50.000, 50.000 and 80.000. Grayscale has been incorperated in raw moments for this instance.

3.3 Selection method conclusion

From all of the selection methods discussed, the user defined pixel intensity boundaries gives the best results. For all encountered characters there is always a possible boundary set which selects the character perfectly. And most of the time no correction needs to be made on the boundaries after the first character is selected. From the automated boundary methods the only one which is remotely practical is based on small image moment change. However this often fails for characters which vary intensity greatly and requires even more user correction than the first method. Therefore the static boundary method is deemed the best for selection in the program.

4 Model construction

The main objective of the proposed method for character recognition is to construct a model using only a few pages of a document that should allow recognition and classification of the rest of the document. Key to the model construction is the extraction of one or more geometric image moments that represent the character as selected by the user. After selection and class assignment by the user, a prototype is created. A geometric moment is used to define the prototype, as it can be transformed into other image moments. This allows switching between several types of image-moments.

4.1 Class/character

A selection is classified to a character class. This character class represents the character that is written to text after recognition. It contains the string that is printed to text when recognized as well as whether the character is italic, bold and/or underlined. The same character for example italic should only differ slightly in shape. But in reality it gives very different image moments due to the use of non-rotational invariant moments. This is why an italic "i" is classified as a different class than a regular "i". Although bold and underline is less prevalent than italic in historic text, being mindful of these types of formatting leaves room for additional formatting marks in the future such as gothic text or capita marking.

Super node

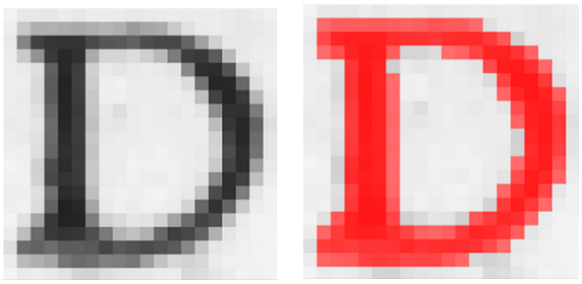
A selection image region that represents a character will always be of many nodes. But nodes are all connected as they are part of the same branch of the tree, and therefore share one or more common ancestor nodes. The node that has no parent in a selection is referred to as a *super node*.

Because there are so many nodes in the selection, making them all prototypes and saving them means a drastic increase in complexity for the recognition part. By use of this method the character illustrated in figure 12 can be reduced from a selection of over 120 nodes to only the single super node.

As mentioned, the key factor of a super node is that it's parent is not in the selected area. Using algorithm 3 the super node(s) can be found in a set of selected nodes.

Algorithm 3 Super node retrieval from selection.

```
for each node in selection-set
  if node.parent NOT IN selection then
    superNodes.add(node)
```



(a) Original character "D". (b) Selected character.

Figure 12: Selection of character "D" consists of a total of over 120 nodes, which has a single super node of the highest grayscale value.

By design a parent of the node has all the information of the children. So a super node contains all the moment data that it's children have. The geometric image moment from the extracted supernode is the image moment that defines the prototype. Only the raw moments are stored so the requested type of moments can be computed later on when loading the model.

4.2 Prototype

The character a user wants to recognize can consist of one or more symbols and as noted before, the character itself can actually produce multiple characters of output. As mentioned previously some character are made up of multiple symbols, these symbols in turn may be reused by other characters. For example the "." in the "i" may be used as the "." to indicate the end of a line or used in abbreviations. The relation between a character, symbols and prototypes is as illustrated in the image below.

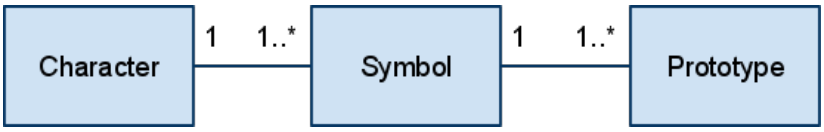


Figure 13: Diagram to illustrate relation between character class, symbols and prototypes.

Looking at this illustration from a recognition standpoint would mean each symbol should first be recognized and later formed into a character. A user would then be trusted with the task to select these symbols, and link them to a character.

From a technical standpoint this method seems sound, yet when taking into consideration the users view of characters, symbols and prototypes; this solution

becomes a lot more complex. The notion of a symbol is difficult to convey to a user. The user would be required to create symbol classes for each of the selected nodes which is not desirable.

To avoid the usability mess, prototypes are made directly for characters, ignoring the intermediate symbol step. The illustration below shows the new relation between character classes and prototypes.

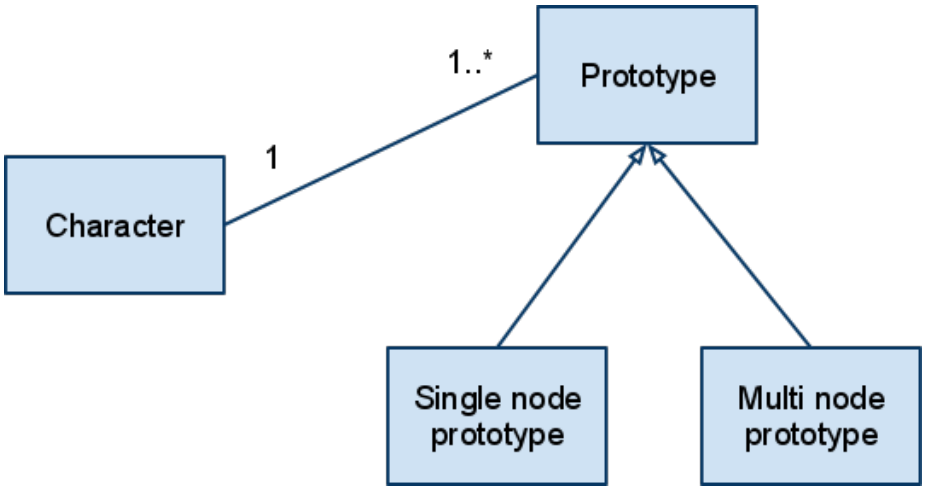


Figure 14: Diagram of simplified relation between character class and prototypes.

The problem for recognizing multi node characters is shifted from recognizing intermediate symbols to recognition of single or multi node prototypes directly during recognition.

Single node prototypes store the geometric moment of the found super node. These geometric moments are already computed during Max-Tree construction. From the geometric moments other moments can be calculated when needed for which more valued attributes as discussed in the Image moment section are valid. The relation between a single node prototype and it's nodes is now that simply only a single geometric moment is stored to represent that node as illustrated below.

Figure 15: Diagram of relation between single node prototype and the geometric moment which is used to describe that prototype.

4.3 Multiple node prototype

As mentioned before the most difficult characters to prototype are a combination of multiple nodes. There are two forms of characters that contain multiple nodes.

The first is when a character, for example an “s” has faded due to document degradation to such a degree that it is broken. Mind that the old “s” character looks like a “f”.

The second type is a diacritic character like “ü” or multiple-part character like an “i”.

(a) Original character. (b) Bottom part selected. (c) Top part selected.

Figure 16: Broken character “s”

4.3.1 Broken characters

In case of a broken character that now has 2 distinct segments, there are already prototypes that of only 1 supernode. Naturally the shape of the two segments differ much from the prototypes already made of “s”. So when the image is processed for recognition, the broken character is recognised in two parts, one as an “l” and one as a dot or an “o”.

A solution to this problem can be found by selecting the largest node and increasing the distance between the lower and upperbound levels. Sometimes this is sufficient to select both nodes when proximity is high and the two nodes connect at some higher level which is not high enough to add additional pixels. When this is successful the broken “s” is then recognized because of the new prototype.

Unfortunately this does not work in the image of figure 16. The “s” is broken into the sections of images 16b and 16c. The connection between the segments is too light of intensity. Too light means that it is of the same level as the background. If selected through the above method, the floodselect algorithm would select a lot more background, making the selection unusable as a prototype.

4.3.2 Multiple segments

The same problem as above occurs with natural multi-part characters, as there is no connection between the characters to make them recognised as one node. Looking at how flooding is done and how 6/8 neighbouring works it also very logical that these characters are not connected, as they are not neighbours.

4.3.3 Developed Solution, multi-select

A solution developed to recognise both was done as followed, by using shift-selection multiple nodes could be selected. This works with both broken or multiple segments characters. The way it works is that when holding shift additional segments can be selected. All selections use the same selection-lowerbound/higherbound value. When saving these supernodes as a Multiple Node Prototype an image moment is created of the combination of the segments by summing the invididual raw moments.

When this new raw moment is created it's centroid is compared to the each segments centroid to determine which super-node is closest.

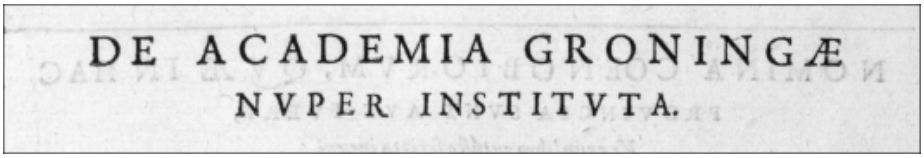
For example with the "i" there are two segments and thus two super-nodes.

When adding the raw moments of these supernodes, the new raw moment will have a centroid closest to the stem of the "i". For further reference this will be called the main or master segment super-node.

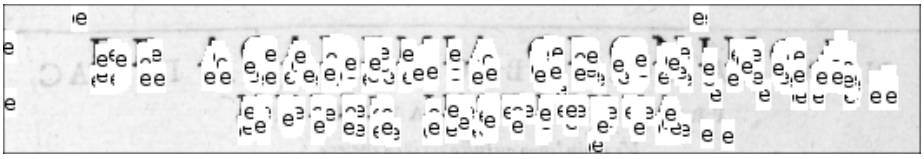
Then a vector and distance is calculated between the main-segment and every other segment. This information is then used in the recognition process.

5 Recognition

Recognition is in broad terms a two stage process. Every node is matched to a single or multiple node prototype. However remember that each node is commulative per level. Which means that there are too many recognized characters, and a lot of nodes barely even resemble their best matching character. The resulting recognized set can be seen in figure 17.



(a) Input image.



(b) Recognized characters at their position. Total of 2852 characters recognized.

(c) Recognized characters of collapse.

Figure 17: Recognition of character "E" from one single node prototype "E" for all nodes.

Clearly some form of post-processing must be performed to reduce the number of recognized characters and filter bad matching nodes. This large set of recognized nodes are reduced in the second stage which is called the "collapse" step. It collapses all recognized characters to just a few making use of the parent/child relationship.

5.1 Single node recognition

The first stage of recognition is matching a node from the dataset to one of the prototypes available. At this point prototypes have been established and hold one vector of image moments and a label refering to a character. In addition it also holds the upper and lower bound levels used in selection.

5.1.1 K Nearest Neighbours

The distance metrics play a key role in comparing a prototype and a vector. the distance calculated between the two can then be stored and used to compare to other prototypes to determine the final prototype.

For this reason K nearest neighbours was implemented. K nearest neighbours works as is traditional also in our case by calculating the distance of all prototypes to each node in the data set.

The distances are stored in a vector where they are sorted in ascending order.

A K-value is defined by the user in the program which is used to determine how many neighbours of the same class must exist before it is recognized. When a K-value of 1 is selected, the prototype that is closest to the data-node is selected as the winner. With K-value defined as 3 the three closest prototypes are selected and it is calculated which class or character in this case is more frequent.

When a character label has been selected the node is labeled, and the first step is done.

5.1.2 Other learning algorithms

k-NN is a type of instance-based learning, or lazy learning, where the user is expected to provide a dataset of truths. LVQ is very similiar to k-NN, however prototypes are moved towards or distanced from the datapoint if classified correctly or incorrectly. It is therefor supervised and the program could ask for user input asking if this is correct or even helping it change, very similiar to T. van Laarhoven's demonstration program [12]. Unfortunately there was not enough time to implement other learning algorithms.

5.1.3 Prototype grey-levels within range.

Prototypes hold lower and upperbound grey levels which were recorded when saving the prototype during selection. Consequently during recognition if the option in the GUI is checked, the data-points level is used to determine which prototypes are candidates for comparison by checking if the level of the datapoint is within the level rang of the prototype. This reduces alot of computation, and avoids that higher level blobs and noise are recognized.

5.2 Multiple node recognition

When recognizing a multiple node prototype, it is handled in the same way as a normal single node prototype. The main segment is compared to the node being processed. Then the other segment vectors in the prototype are used in an attempt to find the same segments amongst the data-points. From the found nodes merged image moments are created to be compared against the already stored moment in the prototype. Through this translations and transformations such as unequal scaling of nodes do not get recognized as a proper multiple node character. For example with the "i" where the "**.**" is on the left rather that the top of the "l". Or an "i" where the "**.**" is disproporiatenly larger than the "l".

The method is tested with a small image with around 20 different gray-levels comprised of several dot-less "i"'s and real **"i"**'s some larger or smaller than normal, including cases where the dot of the "i" is very large when the stem is very small. Recognition of the multiple node character succeeds for scaled variants and fails, as desired, on incorrect relative positions between nodes and on inequal scaling across nodes.

However the method fails in two ways. The first being it is very expensive to search for other other components at a certain position per prototype per datapoint. And calculating a new temporary moment also has a serious impact on performance. Also the method does not work well when having to compete against many of single-node prototypes.

5.2.1 Different approach

A different solution to this problem is to make prototypes per segment. For example a Multi-Node-Prototype would be made consisting of two single node prototypes the dot of the i and the stem of the i. The user could then select and select these "sub-prototypes". Then simply after making the individual prototypes they should be able to bond them and give the combined prototype the correct label the "i".

In the recognition process nothing would be changed, thus not slowing down the recognition like before.

After collapsing, the node with "i-stem" and "dot" would be combined to "i" through use of a vector and distance.

The idea was tested by individually recognizing diacritic characters by the dot and the stem. This resulted in positive results however did not make it in the final design of the program.

5.2.2 Desired solution

The best solution is also the most complicated one. Instead of flooding and defining connected components by neighbouring pixels of the same level, it is more desireable to segment the page beforehand in regions of possible characters.

This should be done during construction and require document analysis to actually work. However if there is an algorithm that can segment a page into characters correctly, each node would then contain multiple segments but would

still be part of the same node. When selecting a character it would also be easier, as a character could be selected by selecting the region around it.

This new function would however require a major overhaul in the program's code and so this feature is deemed as something that should be expanded in the future.

With the user defined set of characters text recognition is performed. This process starts with selecting the best matching characters for all nodes in the Max-Tree. In some cases the node resembles a part of a multi node character which start a context recognition.

5.3 Collapse

After all nodes are matched to their best matching prototype, a character is recognized not only on it's base level, but on many levels individually. All these recognized characters need to be reduced to just a few characters on their most accurately matched level. For this tasks the collapse algorithm is used. The name is derived from the concept that a Tree collapses and only the nodes that solidly recognize a character remain upright. The rest of the nodes recognized character are removed as well as duplicates. The result is a none overlapping set of characters which should be the best representation of that character.

The algorithm starts at every leaf of the tree and traverses each path to the root. When a stop condition is encountered the best recognized character in the active path is selected based on some criteria discussed in 5.5. The traversal algorithm is discussed first.

Tree traversal

The tree is traversed from leaves to the root. For each leaf it's path to the root is followed until either the root or already collapsed node is encountered.

Node- and recursion state

To determine whether a node has already been encountered a node's traverse state is stored. This state can be one of three values as seen in the list below. Initially all nodes are set to "NOT_COLLAPSED". All nodes which are traversed and have a best node in their path are labelled either "COLLAPSED" or "COLLAPSED_AND_RECOGNIZED" depending on their position in regards to the best node. The nodes closer the root are labeled "COLLAPSED" whereas the nodes towards the leaves are labeled "COLLAPSED_AND_RECOGNIZED".

```
enumeration TraverseState {
        NOT_COLLAPSED,
        COLLAPSED,
        COLLAPSED_AND_RECOGNIZED
}
```

The need to distinguish between "COLLAPSED" and "COLLAPSED AND RECOGNIZED" is because when the first is encountered a character may very

well still be recognized in the path, while the latter means that the sub tree already contains a recognized character.

To indicate how a node should be labelled a recursion state is passed from the recursion function of the parent node. These states can be any of the states seen in the list below.

```
enumeration RecurseReturn {
        NO_BEST_NODE,
        WAIT_FOR_BEST,
        RECOGNIZED
}
```

When "NO_BEST_NODE" is returned to the function caller the node should be labelled "NOT_COLLAPSED". This occurs when no best node is found in path, either because no recognized characters are in the path or when the best node does not meet a noise criteria.

The other two states are used to indicate where the node at hand is positioned relative to the best node in the path. When "WAIT_FOR_BEST" is returned the node should check whether it is the best node. If not so it state is set to "COLLAPSED". If it is the best node in the path it state is set to "COLLAPSED_AND_RECOGNIZED" and "RECOGNIZED" is passed on to it's function caller. When a node receives "RECOGNIZED" from it's function callee it simply sets it state to "COLLAPSED_AND_RECOGNIZED".

5.4 Collapse Algorithm

In this section the actual collapse algorithm traversal is discussed. The collapse make a smaller subset of recognized characters from it's starting state by removing nodes recognized character. It consists of two section the first which is done once for the tree which calls all leaves. The second part is recursive node function which does the actual work.

The code in algorithm 4 shows the initial call to a Max-Tree to collapse it's leaves.

Algorithm 4 Collapse tree.

```
Tree::collapseTree()
  sort(leaves, darkest_nodes_first)
  for every node in nodes
    leaf.traverseState = NOT_COLLAPSED;
    for every leaf in leaves
       bestNode = NONE;
     leaf.recursiveCollapse(bestNode);
```

For each leaf node the path to the root is traversed. The algorithm is such that leaves processed first have a higher chance to encounter unrecognized paths than leaves processed later. Only when encountering an unrecognized node a character can persist in the path, so these earlier leaves have a bigger chance

of having a recognized character. Since dark leaf nodes are likely to be a node inside a character these will be handled first by sorting darkest nodes first.

After sorting the leaves, darkest first, for each leaf the recursive procedure as seen in algorithm 5 is started. This function consists of four parts. The first part are the recursion base conditions. Next is the best node selection method discussed later in section 5.5. The third recurses to the parent. And finally based on the parent return value the node updated.

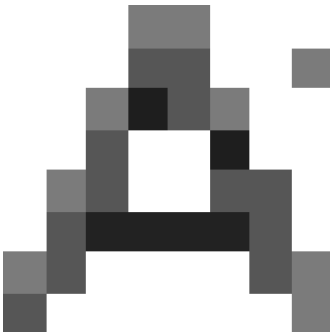
Algorithm 5 Recursive collapse node.

```
Node::recursiveCollapse(bestNode)
  if traverseState == COLLAPSED or this == ROOT
      traverseState = COLLAPSED; character = NONE;
    if bestNode == NONE
      return NO_BEST_NODE;
    if bestNode == isNoise()
      bestNode = NONE; return NO_BEST_NODE;
    else
        return WAIT_FOR_BEST_NODE;
    if traverseState == RECOGNIZED
      bestNode = NONE; character = NONE; return RECOGNIZED

  updateBestNode()

    action = this.parent.recursiveCollapse(bestNode)

if action == RECOGNIZED
  character = NONE
  traverseState = RECOGNIZED
else if action == NO_BEST_NODE
  character = NONE
  traverseState = NOT_COLLAPSED
else if action ==  WAIT_FOR_BEST_NODE
  if this == bestNode
    traverseState  RECOGNIZED
    action = RECOGNIZED
  else
    character = NONE
    traverseState = COLLAPSED
end if
return action
Node::isNoise()
  return similarity > noise_boundary OR area < area_boundary
```

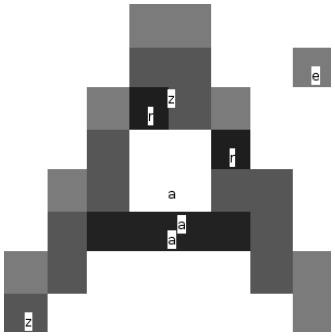
First of the algorithm must stop traversing when either root or a "COLLAPSED" node is encountered. If no best node is found or the best node noise all nodes in path are simply cleared. When best node is valid the path is labelled "COLLAPSED" until node matches best node after which nodes are labelled "COLLAPSED_AND_RECOGNIZED". This is done by returning either "WAIT_FOR_BEST_NODE" or "RECOGNIZED" respectively. All nodes in the path except the best node are cleared of their recognized character. If an already node labelled "RECOGNIZED" is encountered, the traversed path is labelled RECOGNIZED as well and all nodes recognized character.

Example

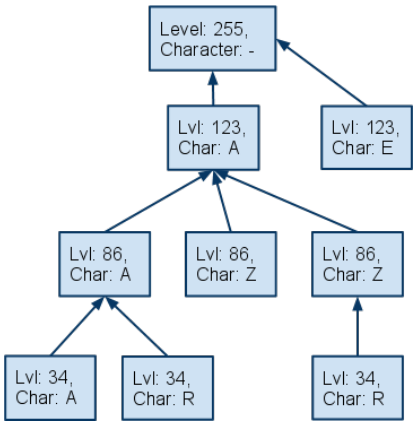
The collapse traversal algorithm will be further explained by using the running example seen in figure 18.



(a) Character "A" input image.



(b) Recognized characters positioned at center of recognized node before collapse.



(c) Max-Tree of input image.

Figure 18: Example image of character **"A"** after some prototypes have been constructed among which the entire **"A"**.

Let us assume that recognition starts on the leaf marked yellow in figure 19. The path towards the root is traversed until "COLLAPSED" or root is encountered. In the first call the algorithm continues all the way to the root node. At this point a best node is known. Lets assume that the green marked node has been picked as the best node by the best node method. Next step is handling the function returns in all the path descended nodes.

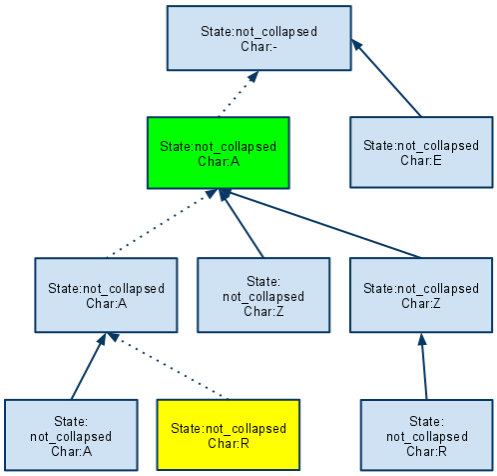


Figure 19: Collapse function traverse to root from the leaf marked yellow. The best node in the path is marked green.

After traversing until base condition is encountered the algorithm returns a function value to all nodes in the path. At first "WAIT_FOR_BEST" is passed to all nodes (figure figure 20). These nodes check if they are the best. In this case the first immediate node is the best. This node retains its character class. Following nodes receive "RECOGNIZED" as return state by which they reset their character class and set their state to "COLLAPSED_AND_RECOGNIZED".

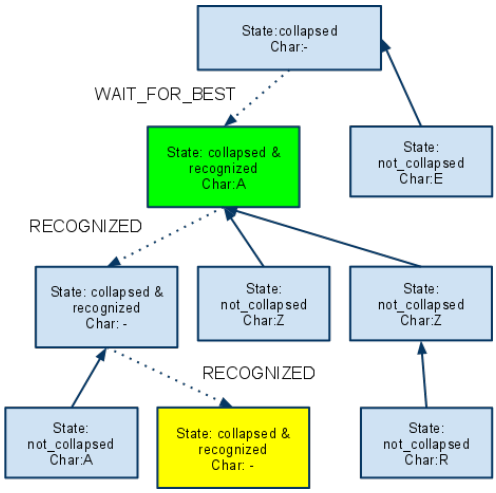


Figure 20: Collapse path function returns. Dotted path return path, text near arrows is return state.

At this point an entire path has been collapsed. Collapse for the next leaf is started. This node travels through it's parents until it encounters a "COLLAPSED_AND_RECOGNIZED" state (figure figure 21). This indicates that the sub-tree has already been recognized and of such all character classes in the path are reset and the state is also set to "COLLAPSED_AND_RECOGNIZED".

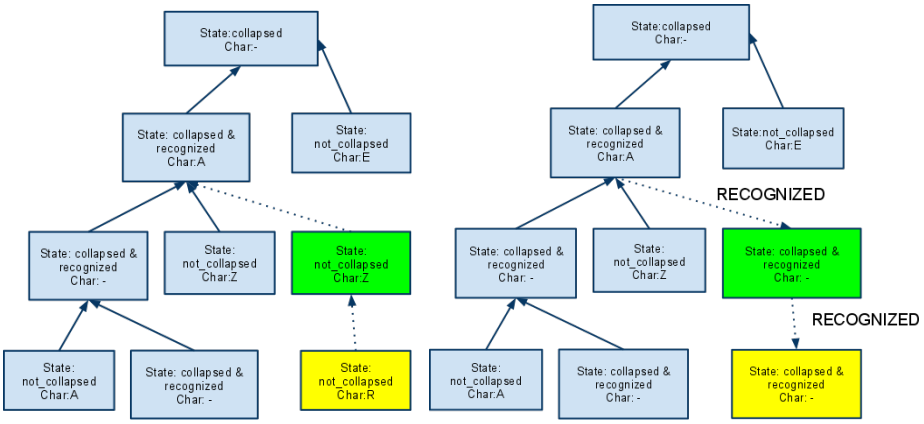


Figure 21: Collapse function from second leaf when encountering already recognized node. Left: call path, Right: return path with return states.

In the same way as in figure 21 two other leaf nodes encounter an already recognized node. After these have been processed the situation is as seen in figure figure 22.

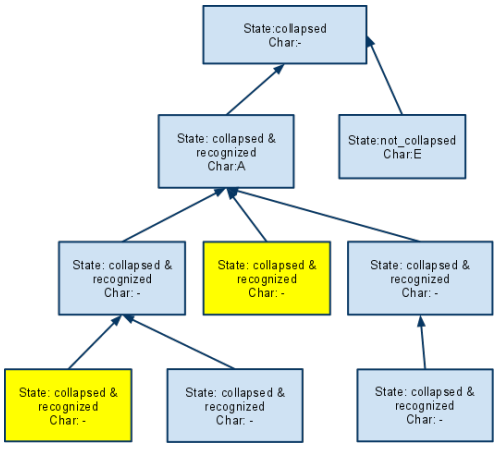


Figure 22: Collapse after two more leaves since figure 21 that eventually encounter a recognized have been collapsed. The newly collapsed leaves are marked yellow.

What remains is one leaf which was clearly a noise pixel in the original image. The path from leaf to root is traversed. After one step a node is encountered and since this node is labelled "COLLAPSED" the path should house a valid recognized character. However since the best node is flagged as noise by the noise function because either it's area is to small, not similar enough to the best matching prototype or some other criteria is not met, the best node is discarded. And all path characters classified are reset.

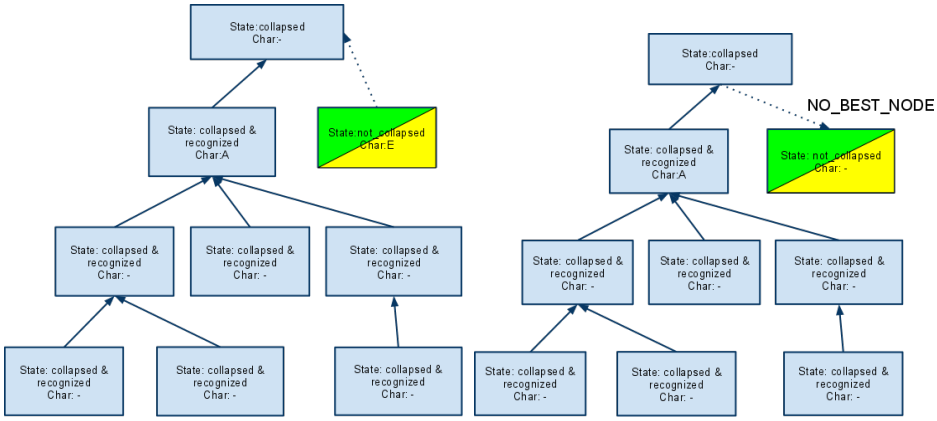


Figure 23: Collapse situation where stop condition allows for recognized character in path, however the best fitting node does is noise by some measurement such as area $<$ N or similarity $<$ epsilon.

After all the leaves are traversed the algorithm is complete and only the "A" on a fairly dark node (intensity 123) remains.

5.5 Best node selection

The method described in the last section only constitutes traversal. The actual heart of the recognition stage lies firstly in how all nodes are recognized, and then how from a path while collapsing the best node is selected. The latter is discussed in this section.

Best node selection happens during or at the end of a path traversal. The best node method is responsible for picking the best character match in the traversed path. The task of the best node selection method is two fold. First, to select the character class which is best represented by the path. Next to select the best node of that class. A few basic winner selection methods have been tested. The first is picking the best matching node in the path. Another method searches for the longest consecutive chain of equal characters.

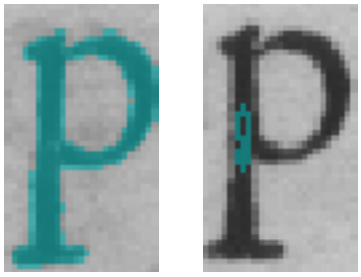
5.5.1 Best matching (lowest distance)

The simplest approach to picking the winning character class and node in a path is selecting the best matching node and it's recognized character as winner. Path 1 in figure figure 25 shows how this method works on a path.

The method works very well, because image moments between characters are very distinct and similar between character instances. A huge downside to this method however is that sub parts of a character sometimes match better than the character as a whole. Such is the case in path 2 on figurefigure 25, where the vertical line (as a lowercase "L") matches better than the total character a node higher which would be a "R".

Figure figure 24a and figure 24b show such an encountered situation where one "P" is recognized correctly and another "P" is seen as an "i".

One source of this issue is scale invariance. A few pixels match just about anything when scaled. Solutions to this problem are discussed later on.



(a) "P" recognized on correct level.

(b) "P" recognized as "I".

Figure 24: Problem in practice where sub node of character is matched before actual character.

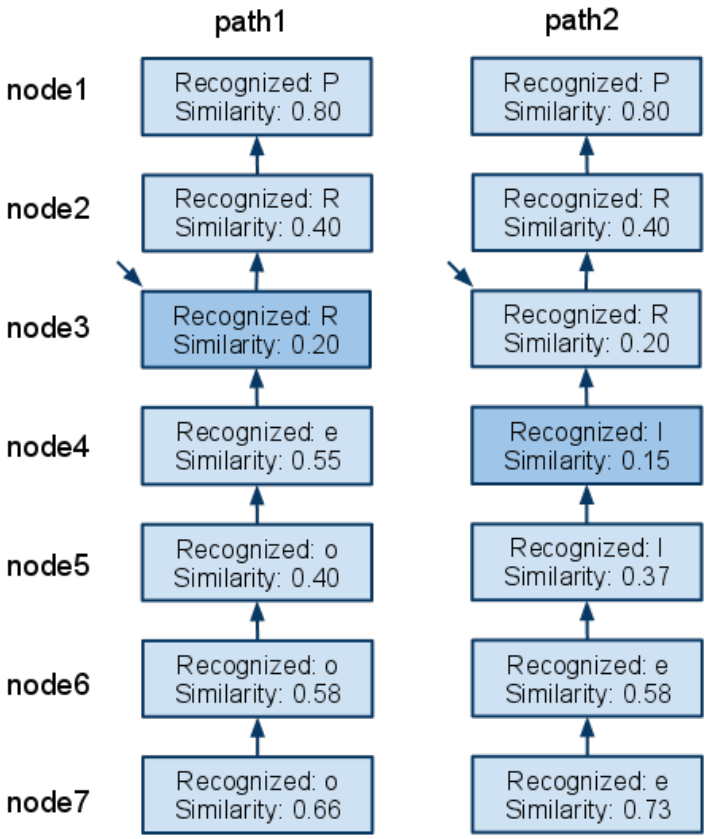


Figure 25: Traversed paths for a hypothetical "R". In path 1 "R" is picked as the best. In path 2 "l" is picked as best matching.

5.5.2 Longest chain

Another method to determine the best node and character class within a path is by looking at character class repetition. The method searches for the longest consecutive chain of characters in a path and selects that character as winner. The node closest to the root within this chain is the corresponding winning node.

Figure 26 illustrates the use of this algorithm. The method often selects the correct character class but wrong node. This is seen in path 1. Most times however either the situation depicted in path 2 or 3 occurs and thus the wrong classification happens. Path 2 is similar to path 2 in figure 24.

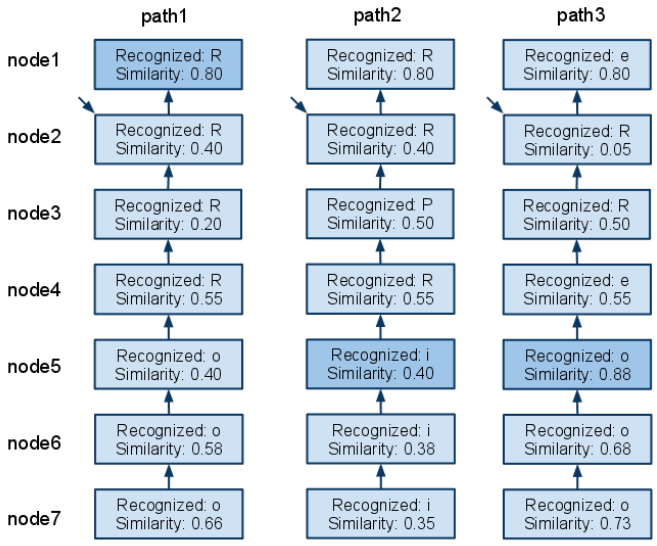


Figure 26: Longest chain best node selection method. Path 1 desired recognition of character, but winning node contains background. Path 2 situation where momentary interruption of chain causes mis classification. Path 3, common ocuring situation where long chain of barely matching classes win for a short chain of accurate matches. The slightly darker nodes and their recognized characters are selected as winners in these paths.The arrow points to the node which actually represent the best character in the paths entire area.

Path 3 is the illustration of another often occurring situation where long set of barely matching nodes actually beats the small set of good matching nodes. The cause of this is that the path of a character is usually made up of say 200 nodes. Roughly 180 of these will represent badly recognize characters. Only the last 20 or so accurately represent the character. This is because on the character boundary pixel intensity quickly drops resulting in less nodes than a small changing gray value section.

Overall this method yields poor results even on clearly printed characters with very little noise. It is of no use on actual historical documents.

5.5.3 Summary

The best matching method performs very well. It however occasionally recognized a part of a character as a better matching character than the whole character. The longest chain method has too many problem scenarios to make it of use in practical situations. Only two different best node selection methods have been tested. The first method discussed gives the best end-result and does not need immediate replacement. Finally, a combination of the longest chain of high simmilarity may yield an even better result.

5.6 Complexity

The collapse algorithm traverses from to leaves to root until an already processed node is encountered. This means that the best complexity is equal to the number of nodes so: $O(N)$, where N is the number of nodes. Worst case already processed nodes turned out te be nodes which means they can be traversed again. The worst case complexity is than traversing from each leaf all the way to the root thus: $O(255 * L)$ where L is the number of leaves. This order may seem less but is larger than $O(N)$ by a factor which will be in the very high digits. In practise only few paths turn out to be noise and thus the order lies close to $O(N)$, or a small multiple of that.

5.7 Recognized character to text

The main focus of this study has been recognition of characters. However for testing and debugging purposes an additional step is done after collapse which is outputting the characters recognized to text. The following rudementary method is used.

First, sort all recognized characters from top to the bottom of the page using the centroids. Then for every two characters the difference in y-space is matched to a certain threshold. If this threshold is exceeded a newline is made.

For each line a similar approach is used to split the line up into words using horizontal sorting.

This method works for a document containing straight lines and similar proportion characters. It fails dramatically however when characters differ in size and when lines are curved. Also the method can not detect multiple new lines, spaces or tabs for that matter. For simple testing and debugging the method is sufficient. For high-scale implementation, an other method such as T. van Laarhoven's proposed line splitting technique are preferred. [12]

6 Findings

Up until this point, individual elements of the proposed ocr method were explained. In this section the method as a whole is reviewed and demonstrated. Following is a test-case with a clearly printed text, after which a test-case using a historical printed document.

6.1 Clear printed text

Given an optimal printed text, recognition is expected to be close to 100% accuracy. The text used is a digital copy of "*As you like it*", a play by William Shakespeare. The font is not that different from the base font in [3].

All pages contain:

- Consistent characters
- Uppercase and lowercase characters
- No noise or distortions
- Equal resolution, characters of equal class are sized equally.

For the test on the "*As you like it*" document the follwing settings are used:

- Classification: KNN is used with k = 1, 3 or 5.
- Image moment: Central moments is used of order 5.
- Distance measure: element euclidian.
- Noise threshold: epsilon of 0.2.
- Collapse best node selection method: Lowest similiarity.

The character set model is constructed using one page of the "As you like it" script. Then another page is recognized based on the model made from the previous page. Multi-node recognition does not work well, so multi node characters and diacritic characters (ë) are classified for each symbol seperately.

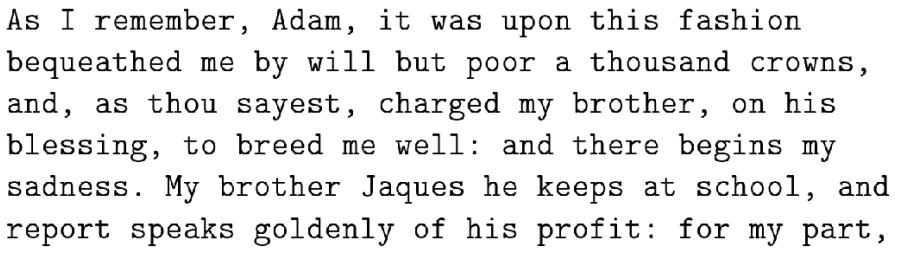
6.1.1 Trained page

In figure 27 the results of recognition from the earlier mentioned settings are applied. From this, the following observations and conclusies are made after model construction and recognition:

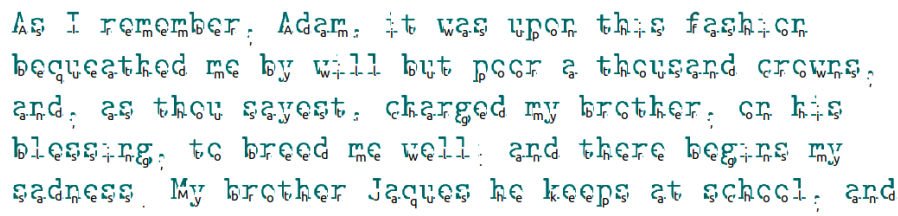
- The image-moments method for classification is prone to make errors in case of similar looking characters. Characters that are frequently mismatched are the following:
  - 'm' <-> 'n'
  - 'v' <->'w'

  - 'k' <-> 'h' <-> 'b'

- The character 'o' is highly problematic. The character is easily misclassified as the resulting central moment is similar to that of many other characters, namely the "D".
- The longest consecutive best node selection method during collapse leads to many misclassifications. The method does however classify well in cases of problematic characters such as those mentioned above.
- Central moments are quite sensitive to minor deviations. A difference of a few pixels drastically affects the central moments, this has an impact on the number of required prototypes. This observation is in line with observations from El Khaly and Maher that character thickness hugely influences recognition [2]. As said, problematic characters require more prototypes in order to properly recognize any deviating instances of those characters. Scale invariant image-moments are less affected by deviating shapes, but in practise perform poorly when used in combination with the proposed classification method, and thus are not suitable at all.
- Simple matching of the node nearest prototype (effectively KNN with k = 1) performs well, while higher values for k do so far less. As the proposed method includes much user-feedback, an imbalance in the number of prototypes per class is bound to be created. This imbalance in the available prototypes leads to misclassification with higher values for k, as the closest prototype is no longer the default winner. Simple KNN does not solve the problem of misclassification of similar characters, but more sophisticated learning algorithms may perform better.



(a) Input image - greyscale and not thresholded.



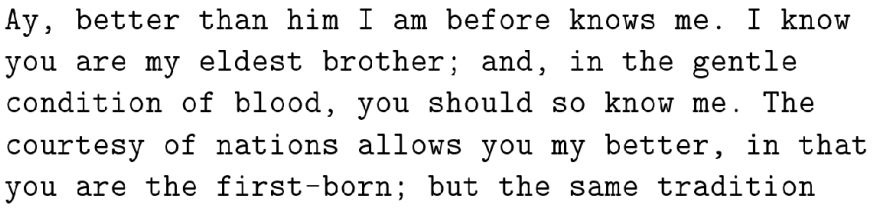
(b) Highlighted nodes in green with recognized characters overlayed

Figure 27: Recognition using central image moments and knn with k = 1 after character model is extended untill 100% recognition is achieved..

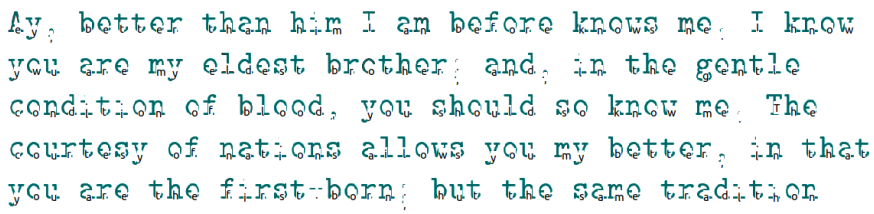
6.1.2 New page

Using the model from section 6.1.1, a new page is recognized. A few lines of this other page and the recognized characters are visible in figure 28.

Seeing as how there is no multiple node recognition the “i” and “;” are recognized as 2 nodes. This gives a total of 204 nodes in figure 28 not counting the root node. Of those 204 nodes a few “m” characters are seen as “n”, and a few severe misclassifations such as recognizing an “e” in an “A”. Finally some errors result from not recognizing the dot of the “i”. The counted number of faults is 13 faults on 204 nodes. This means a recognition 93.6% is achieved. Given that this text is about as optimal input as it can get this is not quite the result that was hoped for. The same problems as in section 6.1.1 are the cause of this.



(a) Input image - greyscale and not thresholded.



(b) Highlighted nodes in green with recognized characters overlayed

Figure 28: Recognition using central image moments and knn with k = 1 with character model trained on other page.

6.2 Historical printed text

One of the testcases of this study is to test whether image moments provide a good means of recognizing text in a historic document where paper degredation is common. For this purpose one of the first pages of the book "Rerum" by Ubbo Emmius is used [3].

An intensive period of learning and prototyping of characters was done and saved to a model. The final model contains 53 character of which a total of 127 instances are represented by prototypes.

Using this model many different setups were tested. Where every image moment type provided more or less the same accuracy rating. However normalized central and Hu invariant image moments proved the most effective in recognition.

An example of the settings used, along with a third setting, which shows unexpected behaviour can be seen in figure 29.

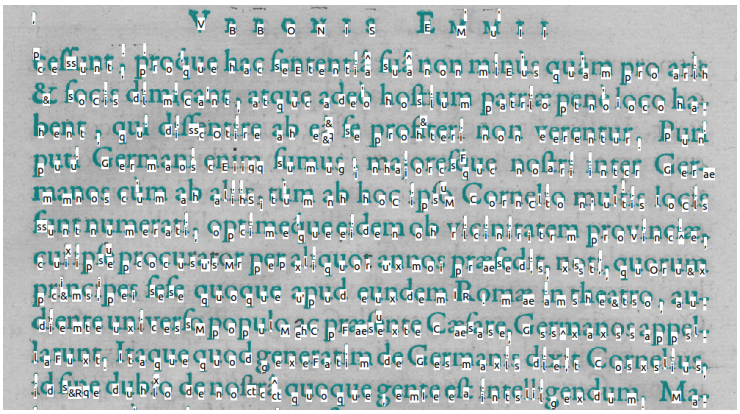
In figure29a the results from the use of normalized central moments are shown which are regarded as the best results after tweaking all parameters. It can be seen that still many misclassifications occur. Comparted to the clear printed text many more errors are made. Many of the same problems as discussed in section 6.1.1 are at play here. For example with the clear text test minor changes in letters cause misclassifications. During testing it is observed that this problem is more prevalent in this document where there is less contrast between a character and the background. Also more prototypes are needed to accurately represent some problematic characters. A characters such as "x" was in need of so many prototypes, that in the end a bias was created to matching

a “x” instead of for example an “a”.

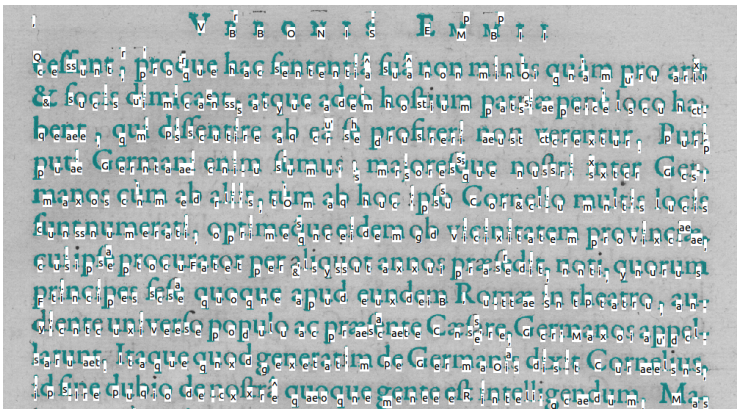
An additional problem with the old document is recognizing only characters, but not noise as characters. It is possible to eliminate background noise lowering the epsilon, which is a threshold where anything above it is marked as noise. This works because only characters as introduced by the user are recognized by the program, and background noise is in theory not part of the constructed model. Using really low epsilon values however is not practical, because in practise it results merely in the recognition of the exact characters the user selected to create a protoype. Therefore a higher epsilon value is used with the consequence that a lot of noise is also recognized as a character. In the future a possible solution would be to be able to classify these noise spots and classify them as garbage.

For comparison the next best image moment type Hu invariant is shown in figure 29b. This moment type performs remarkably well, in some cases even better than normalized central moments. An aspect which however keeps Hu invariant moments from recognizing consistently is it’s rotation invariance. Many characters look similar when rotation invariance is used. A “p” equals a “d” and to some extent a “q” and even less obvious equivalences like a “B” looking like a “m” occur.

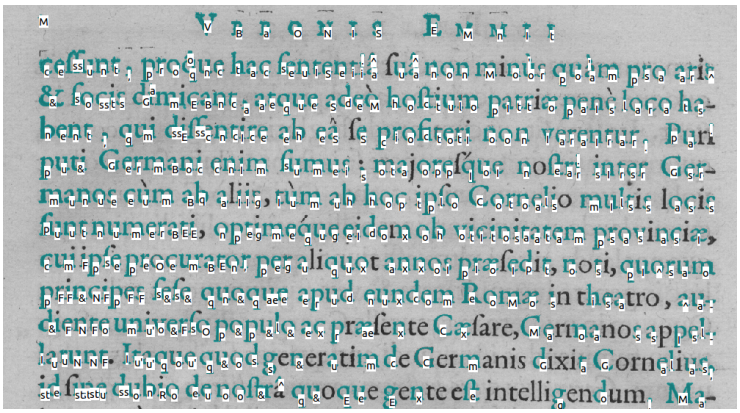
Finally while constructing and testing different parameter settings some unexpected behaviour is observed. In figure 29c it seems the position of the character is a factor during recognition while normalized central moments should be translational invariant. Hu invariant moments are calculated differently from central moments and do not seem to suffer from this. Even though the normalized central moment should be translation invariant, this observation along with the fact that Hu invariant moment performs almost better leads to the idea that perhaps somehow a bug has persisted in a crucial element of the process. After a significant time of debugging the reason for this problem was not found, so it could be that this problem is related to the inner workings of central moments.



(a) Normalized central moment, element euclidian distance, best fit node selection and epsilon of 0.20.



(b) Hu invariant moment, element euclidian distance, best fit node selection and epsilon of 0.20.



(c) Normalized central moment, euclidian distance, best fit node selection and epsilon of 0.20.

Figure 29: Emmius, page 6 recognition using different image moments and distance measures.

6.3 Method Conclusion

This study has tried to answer the research question is posed in the introduction:

> To what extent do image moments provide good character recognition on Max-Tree represented greyscale printed documents?

Along-side this question throughout this thesis another goal has been the minimization of traditional thresholding thus retain as much detail as long as possible through the OCR process. The usage of a Max-Tree in this case has been proven to be a valuable asset, making thresholding not neccesary at first glance as it was proven capable of holding all information of a large historical document along with image moments and additional attributes per node.

The use of image moments as a tool for comparing subsets of pixels works well in theory and in practise on a few distinct characters with little noise. However on large complex historical documents, the amount of noise and differences in shape of characters prove too much to handle. Increasing the order of precision on image moments adds more information and increases differences between character-shapes which makes recognition more precise albeit also more sensitive to small changes. Also higher orders than an order of 5 ($p + q = 5$), used throughout the study, greatly increased computation time, making higher orders not usable in practical application.

Noise recognition problems were tackled by adding rules during the recognition stage like "within level" prototype picking. And during the collapse stage eliminating small sized nodes. Small and large garbage nodes are largely filtered out of the recognition result.

Characters in historic documents are however often recognized as the same, for example an "m" is often recognized as an "n". This is because the largely used normalized central image moment between these difficult characters is very similar. In some of these cases other image moments proved did work, yet none of the discussed moments really sufficed in all situations.

A usability milestone is the selection of nodes in Max-Tree's by using iterative flooding. This proved a good way to select characters without too much difficulty for the user. In the same category; creation, import and export of a model with many prototypes is easy to use.

Despite small recognition setbacks, the combination of Max-Tree's and Image moments for use in the field of OCR has been shown to have great potential. Additional learning methods may further increase accuracy results to a level that it rivals traditonal font-based OCR methods.

7 Future work

The proposed method in this thesis can be improved in a number of ways.

Max-Tree construction can be done differently. Using a concurrent approach Max-Tree construction can be sped up [18]. A method must be used to accomodate the merging of calculated geometric moments for nodes using this concurrent way.

Another change in max-tree construction to assist multiple node recognition is to use define word and character boundaries to make up nodes instead of being defined by a 4- or 8 neighbour function. This is one way to tackle the problem of multiple node recognition such as discussed in section 5.2.2.

A third improvement which also would tackle the multiple node recognition problem is proper splitting of character and symbol such as discussed in section 4.2.

The proposed method of this study can very well also be applied in the use of recognition of a non alphabet character set. Non-latin languages such as hyroglyphs, chinese, japanese and arabic, where font based OCR methods usually have trouble.

Another improvement is the incorporation of image assisted recognition with other OCR elements. For example in the case of 'rerum', the certain characters are deteriorated by high noise levels, e.g. a character 'm' is split in two parts, leaving even a human reader in doubt whether the character is an 'ni' , 'in' or a distorted 'm'. In such cases individual character recognition is no longer adequate, and such (arguably) rare scenarios classification can be perfected by means of a dictionary/lexicon. Besides the use of a lexicon, preproccesing can be used to reduce noise or to further correct for skewness.

Finally the collapse method can be improved by traversing not from leaves to the root, but from the root to the leaves. Although the current method works very well a method which is not biased towards which node is traversed first is preferable.

References

[1] F. Bracci. Automatic traffic sign recognition. Master's thesis, Rijksuniversiteit Groningen, 2010.

[2] Fatma El-Khaly and Maher A. Sid-Ahmed. Machine recognition of optically captured machine printed arabic text. *Pattern Recognition*, 23(11):1207–1214, 1990.

[3] U. Emmius. *Rerum Frisicarum historia, autore Ubbone Emmio,... distincta in decades sex, quarum postrema nunc primum prodit... Accedunt praeterea de Frisia et republica Frisiorum, inter Flevum et Visurgim flumina, libri aliquot, ab eodem autore conscripti.* apud Ludouicum Elzevirium, 1616.

[4] Jan Flusser and Tomáš Suk. Construction of complete and independent systems of rotation moment invariants. In Nicolai Petkov and Michel Westenberg, editors, *Computer Analysis of Images and Patterns*, volume 2756 of *Lecture Notes in Computer Science*, pages 41–48. Springer Berlin / Heidelberg, 2003. 10.1007/978-3-540-45179-2_6.

[5] Jan Flusser and TomÃ¡s Suk. Pattern recognition by affine moment invariants. *Pattern Recognition*, 26(1):167–174, 1993.

[6] Wim H. Hesselink. Salembier's min-tree algorithm turned into breadth first search. *Inf. Process. Lett.*, 88:225–229, December 2003.

[7] Rose Holley. How good can it get? Analysing and improving OCR accuracy in large scale historic newspaper digitisation programs. *D-Lib Magazine*, 15(3/4), 2009.

[8] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IEEE Transactions on Information Theory*, 8(2):179–187, 1962.

[9] Joab Jackson. Google: 129 million different books have been published.

[10] Ronald Jones. Connected filtering and segmentation using component trees. *Computer Vision and Image Understanding*, 75(3):215 – 228, 1999.

[11] A. Khotanzad and Y.H. Hong. Invariant image recognition by Zernike moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 489–497, 1990.

[12] T. Laarhoven. Text recognition in printed historical documents. Master's thesis, Rijksuniversiteit Groningen, 2010.

[13] P. Salembier, A. Oliveras, and L. Garrido. Antiextensive connected operators for image and sequence processing. *Image Processing, IEEE Transactions on*, 7(4):555 –570, apr 1998.

[14] P. Salembier and J. Serra. Flat zones filtering, connected operators, and filters by reconstruction. *Image Processing, IEEE Transactions on*, 4(8):1153 –1160, aug 1995.

[15] Leonid Taycher. Books of the world stand up and be counted!, August 2010.

[16] E. R. Urbach, N. J. Boersma, and M. H. F. Wilkinson. Vector-attribute filters. pages 95–104, 18-20 April 2005.

[17] Erik R. Urbach, Niek J. Boersma, and Michael H.F. Wilkinson. Vector-attribute filters. *Computational Imaging and Vision*, 30:95–104, 2005.

[18] M.H.F. Wilkinson, H. Gao, W.H. Hesselink, J.E. Jonker, and A. Meijster. Concurrent computation of attribute filters on shared memory parallel machines. *IEEE transactions on pattern analysis and machine intelligence*, pages 1800–1813, 2007.