

TECHNNISCHE UNIVERSITÄT KAISERSLAUTERN  
IMAGE UNDERSTANDING AND PATTERN RECOGNITION GROUP  
DFKI

DIPLOMARBEIT

---

# Shape Matching for Automatic Text Reading in Natural Scenes

---

*vorgelegt von:*  
Marius RENN

*Matrikelnummer:*  
349559

*Gutachter:*  
Prof. Thomas M. BREUEL

*Betreuer:*  
Adrian ULGES

KAISERSLAUTERN  
November, 2008



Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Textauszüge und Grafiken, die sinngemäß oder wörtlich aus veröffentlichten Schriften entnommen wurden, sind durch Referenzen gekennzeichnet.

Kaiserslautern, im November 2008

Marius Renn



## Acknowledgements

First and foremost, I would like to thank Thomas Breuel and the whole IUPR team for providing me with a wonderful learning experience in the field of pattern recognition and image processing. I have been working with the team for a few years now, and value the open atmosphere, friendliness and intelligence of all of the team members.

I would especially like to thank Adrian Ulges for his help, assistance, and patience during the completion of this work. I would also like to thank Christian Schulze for keeping an eye on my work, giving useful tips, and helping out in most of the organizational and paper work.

Also, I would like to thank all of the remaining members of the IUPR for their helpful insights, discussions and tips, especially the whole InVire discussion group, including Damian Borth, Manni Duan, Christian Jansohn, Markus Koch, and others I have forgotten to mention here.

Of course, I would like to give a special thanks to my parents for their constant support on all my endeavors, and for shaping me to the person I am today. I would like to thank my sister and brother, who are by my side whenever I need them. Finally, I would like to thank my wonderful wife for her endless patience, support and kind words that kept me going when times got tough.



# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Outline . . . . .	7
1.2	Traditional OCR . . . . .	8
1.3	Text Reading in Photographs . . . . .	8
1.4	Text Detection and Text Reading . . . . .	10
1.5	Related Work . . . . .	10
1.5.1	Text Reading in Natural Images . . . . .	11
1.5.2	Text Detection in Natural Images . . . . .	12
1.5.3	Text Detection and Reading in Videos . . . . .	13
<b>2</b>	<b>System Overview</b>	<b>15</b>
2.1	Character Hypothesis Generation . . . . .	15
2.2	Filtering . . . . .	17
2.3	Grouping . . . . .	18
<b>3</b>	<b>Character Hypothesis Generation</b>	<b>19</b>
3.1	Approaches . . . . .	19
3.1.1	Separate Detection and Recognition . . . . .	19
3.1.2	Joint Detection and Recognition . . . . .	21
3.2	Preprocessing . . . . .	21
3.2.1	Sobel Operator . . . . .	21
3.2.2	Laplacian of Gaussian . . . . .	22
3.2.3	Canny Edge Detection . . . . .	23
3.2.4	Distance Transform . . . . .	24
3.2.5	Connected Components . . . . .	25
3.2.6	Skeletonization and Thinning . . . . .	25
3.3	Character Detection . . . . .	27
3.3.1	Closed Contours . . . . .	27
3.3.2	Strokes . . . . .	27
3.4	Character Recognition by Feature Classification . . . . .	28
3.4.1	Orientation Histograms . . . . .	29
3.4.2	Fourier Descriptors . . . . .	30
3.4.3	Skeleton Classification . . . . .	32
3.4.4	Classification . . . . .	33
3.5	Character Recognition by Prototype Matching . . . . .	34
3.5.1	Template Matching . . . . .	34
3.5.2	RAST Matching . . . . .	35
3.6	Character Hypothesis Filtering . . . . .	38

<b>4</b>	<b>Grouping</b>	<b>40</b>
<b>5</b>	<b>Experiments and Results</b>	<b>43</b>
5.1	Tests on Synthetic Data . . . . .	43
5.1.1	Character Detection . . . . .	44
5.1.2	Character Recognition on Known Locations . . . . .	50
5.1.3	Character Detection and Recognition . . . . .	60
5.1.4	Summary . . . . .	60
5.2	Word Recognition on Real-World Data . . . . .	61
5.2.1	Summary . . . . .	66
<b>6</b>	<b>Conclusion and Perspectives</b>	<b>68</b>
6.1	Conclusion . . . . .	68
6.2	Outlook . . . . .	68
	<b>References</b>	<b>70</b>



## List of Tables

1	Overview of preprocessing, detection and recognition techniques . . . . .	19
2	Principle axes and orientation histograms . . . . .	30
3	Overview of synthetic image sets . . . . .	45
4	Closed contour performance . . . . .	46
5	Stroke detection performance . . . . .	49
6	Performance of character detection methods . . . . .	49
7	Performance of filtered character detection . . . . .	50
8	Example character samples used for training . . . . .	51
9	Template matching performance . . . . .	51
10	Evaluation of $k$ -nearest neighbor classification . . . . .	52
11	Orientation histogram performance . . . . .	53
12	Fourier descriptor classification performance . . . . .	53
13	Skeleton classification training performance . . . . .	54
14	Skeleton classification testing performance . . . . .	55
15	RAST baseline performance . . . . .	56
16	RAST performance with rotation . . . . .	56
17	RAST performance using adaptive edge detection . . . . .	57
18	RAST performance using rotation, scale and translation . . . . .	58
19	RAST performance trained on multiple fonts . . . . .	58
20	Performance summary (single font) . . . . .	59
21	Performance summary (multiple fonts) . . . . .	59
22	Character detection and classification performance . . . . .	60
23	Joint detection and recognition performance . . . . .	60
24	Computational efficiency of reading methods . . . . .	61
25	Text detection performance on ICDAR . . . . .	62
26	Separate detection and recognition performance on ICDAR . . . . .	63
27	Joint performance on basic subset . . . . .	65
28	Joint performance on difficult subset . . . . .	65
29	Joint performance on noisy subset . . . . .	66

## List of Figures

1	Examples of text occurrences in natural scenery. . . . .	5
2	OCR Processing steps . . . . .	8
3	Examples of text in natural scenery . . . . .	9
4	Steps in the text recognition system. . . . .	16
5	Seperate approach using recognition by feature classification . . . . .	20
6	Sobel edge detector output . . . . .	22
7	LoG edge detector output . . . . .	23
8	Canny edge detector output . . . . .	24
9	Distance transform metrics . . . . .	25
10	Visualization of connected components . . . . .	25
11	Ridge points of the distance map . . . . .	26
12	Raw and cleaned skeletons . . . . .	27
13	Visualization of stroke width/color variance . . . . .	28
14	Visualization of contour orientations . . . . .	29
15	Contour reconstruction using Fourier descriptors . . . . .	31
16	Visualization of the phase . . . . .	31
17	Fourier descriptor normalization . . . . .	32
18	Character skeletons and their discrete properties . . . . .	33
19	Sample template matching . . . . .	36
20	Illustration of geometric matching . . . . .	36
21	Grouping process . . . . .	40
22	Character constraint violations . . . . .	41
23	Examples of challenging synthetic test images. . . . .	44
24	Influence of closed contour parameters . . . . .	47
25	Example of failed detection using closed contours . . . . .	48
26	Comparison of skeleton algorithm output . . . . .	48
27	Influence of stroke detection parameters . . . . .	49
28	Influence of the number of Fourier descriptors . . . . .	54
29	Issues when dealing with rotation . . . . .	57
30	Visualization of RAST matches . . . . .	59
31	Sample ICDAR images . . . . .	62
32	Successful image reading samples . . . . .	64
33	Unsuccessful image reading samples . . . . .	64
34	Examples where joint approach succeeds . . . . .	66
35	Example image from the noisy set . . . . .	67



Figure 1: Examples of text occurrences in natural scenery.

## 1 Introduction

We call the time we live in “the Information Age”, and while we may associate information primarily with computer technologies, such as the internet and large databases, the real world around us is filled with information like never before. Most prominently, information is manifested as text: It is printed on signs, painted on the ground, glowing from advertisements or store signs. It is on our book-shelves, on paper on our desks, on appliances and other products, on packaging, on our clothes, and sometimes even etched into our skin. We suggest the reader to take a look around, and try to spot all occurrences of text around him- or herself. This may be more difficult than it seems: Text has become so ubiquitous that we are often oblivious to its presence. Figure 1 shows a collection of images that give examples of where text typically occurs in the real world.

Although we might be oblivious to most of the text around us, it is not alarming that it goes unnoticed, as much of it carries no information that is relevant to us at that very moment in time. On the other hand, text may carry information that we wish not to ignore: It can be a warning on a sign, the contents of a package or a letter sent from a friend. This is all the more frustrating when we cannot read the text, either due to a deficiency in sight, or the text being in a language we do not understand. This is one of several scenarios where automatic text recognition would come in handy: Instead of looking up word for word in a dictionary (which may be all the more difficult if the characters are not from our native alphabet), the unknown text could simply be photographed with a mobile device, and a text recognition engine would (hopefully) recognize the text and display a translated version. Of course, such a recognition engine would have to be capable of locating and reading text found in random photography. We shall call such a system a “text reading system”:

**Definition:** A *text reading system* is a software system for automatically detecting and reading scene text within an image (usually a photograph).

We have outlined automatic sign translation as one field of application for a text reading system. This is just one of the many scenarios where a such a system would come in handy. Other scenarios include:

- Text reading in large image databases: Many large image databases, such as the popular photo-sharing sites found online<sup>1</sup>, need to organize the image data they aggregate to allow users to search or group images. This is done by associating each image with a textual description of its contents. Often this is done manually by a human (such as the photo contributor), who annotates the image with a set of tags. In other cases, the filename is used as a textual description, and in the worst case, no meaningful description is used at all. However, in a study of 200 random images from the Flickr website, 55% of them were found to contain text. In many cases, the text found in these images would have been a meaningful addition to the textual annotation, and in any case it would have been better than no annotation at all. An automatic text recognition system for random imagery could supplement photo annotations with the text found within the image.
- Video indexing: Just as in photo indexing, text reading could be used to index videos. This could be done on a per-video level, or on a per-scene or per-frame level. For instance, such a system would allow a viewer of a recorded video to jump to the first frame where “Midnight News” occurs.
- Object meta-data: As stated before, many of our everyday products contain text on them, whether it is the name of the product, the brand name, pricing, weight, or ingredients. A text reading system could associate these products with meta-data found on the packaging. Such meta-data could then again be used in an object-recognition system in order to make a decision on which object class it is dealing with. A detected rectangle might not be as meaningful as one supplemented with the detected words “TV Guide”. Object-bound text information could also act as an identifier for certain items. This could be a license plate number of a car, or a serial number on a product.
- Sign reading: In the example above, sign reading was used for translation. However, there are many more potential uses for a sign reading system. For instance, coupled with speech synthesis, signs could be read out-loud for the visually impaired. Together with map information, the text on signs at a street intersection could be used to pinpoint the location of the text recognition device. Sign text could also be used to update status information of a dynamic system, such as a navigation system in a car: The system could warn the driver when certain signs are detected, or update navigation data when an unexpected sign value is read, such as the temporary reduced speed value at a construction site.
- Tracking: Text reading could be used to track objects or people that have a certain word attached to them. For instance, a tracking system could use the player number on a jersey as a feature for tracking team players in sports videos.

Despite remarkable advances in OCR technology in the past forty years, the recognition of text in photographs still poses a great challenge, and conventional OCR systems fail at the task. The reason for this is that many OCR systems make assumptions about the input image that do not hold for general photography. Text reading systems on the other hand must be able to deal with varying text fonts, sizes and textures. Text may lie on complex backgrounds, may be rotated, scaled or undergo perspective distortion. Furthermore, lighting conditions may greatly alter the color of text and produce specular highlights. Section 1.4 deals with these challenges in more detail.

In this work, a generic text reading framework for photographic images will be introduced, which will be used to test various text detection and reading techniques. Also, a novel approach based on geometric matching is introduced. The main contributions of this work are the following:

---

<sup>1</sup>see <http://www.flickr.com> or <http://www.picasa.com>

- Using the proposed text reading framework with a variety of features and classification methods, several techniques are investigated and their performance on detection and recognition of text in photographs evaluated. The goal is to help establish a “best practice” for text reading in natural scenes. Common OCR techniques and lesser known ones are discussed, covering a large domain of image processing and pattern recognition methods, and a quantitative comparison is provided. To our knowledge no such comparison has been performed to date.
- Unlike most contributions in this domain that have performed tests on small-scale datasets (usually provided by the authors), the tests in this work are conducted on a large scale synthetic database of images, which simulate texture and distortions often found in real-world photography. This will allow full control over font, color and distortion, and test robustness of recognition methods against each of these transformations. Furthermore, the recognition system is tested on real-world photographic images, provided by the well-known ICDAR training and test sets of 2003 and 2005 [27, 26].
- Many publications focus on a single subtask of text reading, usually the character detection phase only. The actual text recognition is then either omitted or conducted using standard proprietary OCR software. Here, an analysis for the entire text reading process is provided from start to finish. That is, the input of the system is a random photograph, and the output is a set of words that appear in the image, along with their position and scale. No third-party framework is involved in the process.
- Finally, a novel method of text recognition in random imagery is introduced that applies a powerful geometric matching technique called RAST. This will allow the classification of characters within an image without any prior segmentation, i.e. the character detection and recognition steps are unified into one joint process. This method of joint detection and recognition is evaluated and compared to the performance of text reading using separate detection and recognition steps.

It should be noted that the focus of this work lies not only on tuning a single text-reading approach, but to provide a thorough component-based evaluation.

## 1.1 Outline

In the remaining sections of this chapter we will first give a brief overview of traditional OCR for readers not familiar with the topic. Then a comparison between traditional OCR and text reading in natural imagery is given, and the challenges involved are discussed. Section 1.5 gives a more detailed explanation of how a text detection engine can be extended to a text reading engine, and gives justifications for this extension. This chapter concludes with an overview of related work. Section 2 deals with the architecture of the text reading framework used in this work. After a general overview, the process of generating character hypotheses is explained, followed by an outline of the filtering and grouping modules. Section 3 details the character hypothesis generation process. It discusses the two main approaches used in this work - separate and joint detection and recognition, and details the processing steps involved. The grouping algorithm is discussed in section 4. All of the discussed methods are then tested on synthetic and real-world data in Section 5, which also provides a detailed discussion of the results. Finally, section 6 draws a conclusion of the work conducted here, and provides an outlook for planned future work.

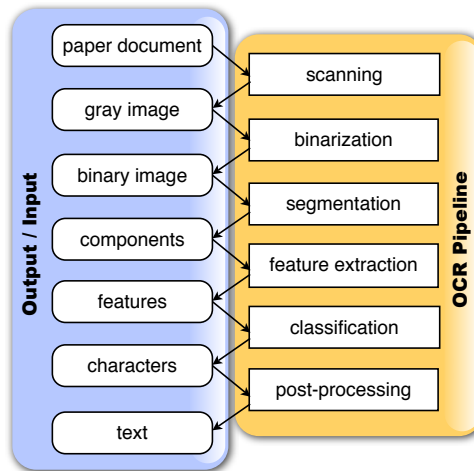


Figure 2: The typical processing steps of a traditional OCR engine.

## 1.2 Traditional OCR

Traditional OCR systems usually make the following assumptions on the text imagery they are dealing with:

- Text is assumed to be dark on a bright background.
- Characters are machine-printed.
- Text is layed-out on straight horizontal lines, that are arranged in text blocks.
- Other than dark text and bright background, nothing is present on the image.

For readers unfamiliar with the text recognition process, a brief overview of the processing steps are given here. Note that these may vary from system to system, but are usually present in at least a similar form.

The input of an OCR image is usually a grayscale or color image, obtained from scanning a document with a (flatbed) scanner. Scanners tend to produce sharper and higher resolution representations of documents than digital cameras, though these are sometimes used instead. Once the image has been digitized, it is binarized, i.e. converted to a black-and-white image, where black pixels are the character pixels, and white pixels are the background. The image is then segmented into single characters, which is usually done by a connected component analysis. From these single characters, features are extracted, that have roughly the same values for representations of the same character. These feature values are then classified to obtain a character class for each component. The components are then grouped to words and sentences. A post-processing step may apply statistical information or grammars to correct misclassifications. Figure 2 shows the typical OCR pipeline incorporating the steps described here.

## 1.3 Text Reading in Photographs

Many of the assumptions made by traditional OCR systems do not apply to text reading systems for photographic images. Obviously, such a system cannot assume that text is black on a bright background. When dealing with text in photographs we must deal with the following challenges:

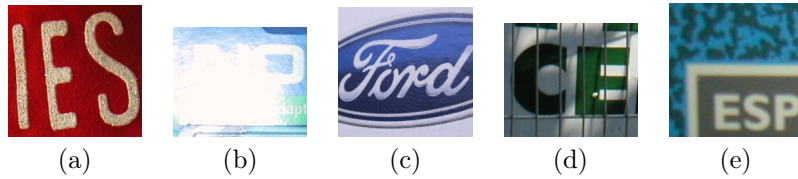


Figure 3: Examples images of text in natural scenery, highlighting a few of the difficulties that arise when dealing with such images.

- Text can appear in any color and texture on a background with any color and texture. Figure 3 (a) shows text with a complex texture on a noisy background. Thus, a binarization step like in traditional OCR that separates text from background is very difficult to accomplish for photographic text. Furthermore, segmenting by connected components will perform poorly with textured characters, or characters that “flow” into the background, as image 3 (b) shows.
- While much of the text encountered in photos is machine-printed, it may be heavily stylized or distorted. Various fonts, untypical for document use, are used frequently in advertisements, product labels and logos. As these text labels should be unique to the brand or product, they are often stylized, altered or use brand-specific fonts. Image 3 (c) shows an example of brand text found in photos.
- While traditional OCR assumes text to be on a flat piece of paper and in frontal view, text in photographs may be rotated in any direction, distorted by the perspective, and lie on curved surfaces.
- Traditional OCR usually deals with high quality image representations obtained from a scanner. A text reading system for photos on the other hand, must deal with noisy images, possibly taken in bad lighting conditions with a hand-held device that produces low-resolution images. Even if the camera is of high quality, the text may be far from the observer, so that the camera captures the text in a small portion of the image only.
- Text in photos usually does not follow any standard layouts. In fact, as text may stem from various sources (such as different signs), their layout may appear random for a recognition system. Still, such a system must be capable of identifying characters and words that belong to the same group, and those that are independent from one another.
- Unlike in scanned documents, text in photos may be partially occluded by another object, as shown in image 3 (d), where the characters are partially occluded by a wired-fence. While the possibility of recognition depends on the scale of occlusion, in minor cases a text reading system should still be able to identify the characters.
- Images may contain a high amount of clutter, that is, many objects appear in the image, that are not text. These objects may out-number the text elements by far. Image 3 (e) shows an image with a high amount of clutter near a text string.

While many challenges presented above are inherent to dealing with text in photos, some challenges are relevant for both traditional and photo text recognition. For instance, both systems may need to deal with so-called linked and split characters. These occur when character data is incorrectly segmented so that either multiple characters are segmented as one (linked), or a single character is segmented as multiple ones (split). One way to avoid these problems is to avoid segmentation altogether. In fact, such methods will be introduced in this work that are robust against these two problems.

Finally, some assumptions are made by a typical text reading system that are more relaxed than the requirements for traditional OCR. Most notably, it is assumed that text in photos consists of a few words only. Simple sentence structure is assumed, and no complex grammar analysis is performed (though a dictionary is used to improve word recognition performance). Often punctuation is simply ignored. The set of recognizable characters may be smaller than that of a document OCR engine. For instance, for sign reading, a text recognition system will normally not require recognition of mathematical symbols.

## 1.4 Text Detection and Text Reading

A common approach to solving the text reading problem is to first detect text within the image, extract the detected text and apply a dewarping operation to remove any perspective distortion. Then, the extracted text is passed to a third-party OCR engine for recognition. In this work however, such an approach is not used. Instead, custom character recognition techniques are implemented, specifically tailored to the task of text recognition in natural imagery. However, with all the difficulties involved in text detection alone, it is appropriate to ask whether such an extension to the more complex task of detection *and* recognition makes sense. After all, why should a well-tested and finely tuned commercial OCR system not be used instead of a custom system developed “from scratch”?

While the additional recognition task does indeed introduce new difficulties, it does bring along a number of advantages that are worth evaluating. Specifically, the following benefits are given from an integrated recognition module:

- The results of the recognition process can be re-used in the detection step. That is, the recognition of certain characters may give rise to additional character bounding boxes. For instance, if a partial string (according to some dictionary or heuristic) is found at a certain location, the detection module could use this information to apply a more detailed search for the missing characters.
- Since generic OCR engines require a binarized, non-distorted text image as input, text found in natural scenery must be prepared for such an engine. This involves segmenting the text string from the background, converting the text to black and white, and eliminating distortion. Each of these processes may introduce error, and a significant amount of information is lost by each of these steps. Hence, here recognition methods are used that adapt to the image data, rather than vice-versa.
- Instead of looking for bounding boxes first and then applying character recognition to each of these boxes, certain recognition techniques can be used on the entire image directly. The detected bounding boxes are then generated along with the character hypotheses (joint detection and recognition).

Some of the character recognition methods presented in this work are related to similar method found in traditional OCR. Others are derived from methods used primarily in shape matching. While these methods may not be able to compete with commercial OCR systems on textual documents, they are far more adequate in dealing with text found in natural imagery.

## 1.5 Related Work

The common method of text reading is to combine the task of character detection with optical character recognition. In the following a selection of prominent work is presented that has been



conducted in the fields of text reading, text detection and geometric matching under bounded error. Here, we will classify a system as a text reading system only if it contains an integrated character recognition module. Although several authors refer to their systems as text reading systems, many of them pass a binarized version of the detected text to a commercial OCR system. As this possibility is available to all text extraction systems, we make a distinction between such systems, and those who use their own end-to-end detection and recognition engine.

An overview over general optical character recognition techniques would be beyond the scope of this work. Readers who are interested in an overview of OCR classification and feature extraction techniques are referred to [32] and [33].

### 1.5.1 Text Reading in Natural Images

Not much research has been conducted in text reading. Most systems focus on text detection only, and pass the detected text image to a general-purpose OCR engine. In fact, although two ICDAR text reading challenges were held, one in 2003, and another in 2005, none of these received a single entry [27, 26]. In the following we will give a brief overview of the prominent candidates in the text reading domain.

The earliest work in text reading was performed by Ohya et al. [31]. The text reading process is performed in 4 stages: First, the image is binarized based on adaptive thresholding. Then, character candidate regions are detected by observing gray-level differences between adjacent regions. A character recognition process selects patterns with high similarities by calculating the similarities between character pattern candidates and a set of standard patterns in a dictionary. Finally, a relaxational operation is applied to update the similarities. While the approach appears to work well for signs, the binarization step is inappropriate where text can show high levels of noise and variations in illumination. Based on experiments on a set of 100 images, the recall rate of text detection was 85.4% and the character recognition rate 66.5%.

Zhang et al. present a system for sign reading and translation [45, 46]. The system is capable of detecting, recognizing and translating chinese signs to English. To detect sign text, a 3-step algorithm is used: The first layer detects possible sign regions by employing a multi-resolution edge detection algorithm. The second layer performs an adaptive search in the neighborhood of the initial candidate regions. The search incorporates layout constraints to restrict the set of possible text regions. The third layer is a more elaborate layout analysis, which the authors claim is essential for the detection of the relatively complex Chinese character structures. Finally, the authors present a character recognition engine in [47] for recognition of text embedded in natural scenes. Instead of passing binary images to an external recognition engine, the presented system extracts features for OCR from the image directly. The character image is preprocessed with an intensity normalization method to deal with lighting. Then, the Gabor wavelet transform is employed to obtain local features over a  $7 \times 7$  grid, and linear discriminant analysis (LDA) is used to reduce their dimensionality. The resulting features are classified to character classes using the  $k$ -nearest neighbor method.

Additionally, we are aware of two commercial text reading system. The SceneReader system is capable of detecting and recognizing alphabetic text in a broad variety of photographic images, including highly complex images such as street scenes, according to the software's website<sup>2</sup>. A visualization of the engine's performance on the ICDAR 2003/2005 datasets can be downloaded from the website. Finally, the software package Shoot & Translate by Linguattec is a text recognition and translation engine for mobile devices<sup>3</sup>. The software is client - server based: After the user

---

<sup>2</sup><http://www.scenereader.com/>

<sup>3</sup><http://www.linguattec.net/products/mtr/shoot/trans>

has taken a photo with the mobile device, the image is uploaded to a recognition and translation server, which sends the translated text back to the mobile device. At this time, the website gives no detailed information on accuracy.

### 1.5.2 Text Detection in Natural Images

Prior work on text detection systems can be roughly divided into those that are based on connected components, and those that make use of textural features. Some of the earliest work in text detection was performed by Lee and Kankanhalli [21], using a connected-component method. An edge detection algorithm is first used to determine gray-level values of pixels on character boundaries. Then gray-level thresholding is performed for connected component generation of objects with the same gray levels. A series of heuristics are used to filter out non-text components, which are based on aspect ratio, contrast histogram, and run length measurement. Though the authors claim the method could be applied to other domains, results are presented only for cargo container images.

Zhong et al. [48] apply a connected component based method using color reduction. The image color space is quantized using peaks in the RGB color histogram. This assumes that text clusters together in this color space, and that they occupy a significant portion of image. The components are then filtered using heuristics, such as area, diameter, and spatial alignment. The authors evaluate their system on a number of images of CD and book covers.

Similarly, Shirai et al. [37] use color-based clustering for component extraction. For this, the image is divided into several blocks, based on detected edges. Each block is clustered in color space to form blobs. An SVM classifier is used to classify the texture of blobs to text or non-text. While the authors state they have tested their system on the ICDAR2003 data set, no results are given.

Jain and Yu [16] apply a connected component based method, which includes color bit dropping (24 to 6-bit) the image, followed by color clustering. The generated color regions are decomposed in multiple foreground images. The connected components are extracted from each of the images and classified. The localized text components are then merged into a single output image. The system is capable of handling non-skewed horizontal and vertical text only.

The text detection method presented by Messelodi and Modena [29] consists of 3 steps: First, elementary objects are extracted from the image, obtained after intensity normalization, binarization and connected component generation. In a second step, these are filtered based on area, relative size, aspect ratio, density and contrast. Finally, text lines are extracted based on character closeness, alignment, and comparable height.

Myers et al. present a system capable of dealing with 3-dimensional text in natural scenery [30]. The text detection and location process detects vertically oriented edge transitions and connected components of similar intensity in grayscale. From these components, horizontal and vertical bounding lines are estimated. The resulting rectangles are then normalized to frontal view, that is any perspective distortion is removed. However, the sequence of text in the image is assumed to be of similar intensity, and the in-plane rotation roughly zero.

Hasan and Karam [14] present a text detection system using morphological operators. After the intensity of the image has been computed, edges are identified using a morphological gradient operator. These are thresholded to obtain a binary image. Spatially close edges are grouped by dilation, while small components are removed by erosion. The components are then filtered based on size, thickness, aspect ratio, and gray-level homogeneity. While performance is discussed, the authors compare their results to other systems on three images only.

While the presented text detection systems so far have based their decisions on connected component and edge features, other text detection systems classify regions into text and non-text using

textural features. Wu et al. [43, 44] present a system called “TextFinder”, which begins by segmenting the image using a multi-scale texture segmentation scheme based on nine second-order Gaussian derivatives. A non-linear transformation is applied to each filtered image to obtain a feature vector for each pixel. These feature vectors are then classified to segment the textures into different classes. The authors refer to this process as *texture segmentation*. Next, the *chip generation* stage is initiated that operates on the segmented regions and applies a set of appropriate constraints to find text strings within these regions. Chip generation consists of 5 phases: The stroke generation phase, in which the input image is convolved by a second order Gaussian derivative in the horizontal direction to find strong vertical edges. These edges are thresholded and grouped to connected components called *strokes*. Then, stroke filtering is used to eliminate false positives, followed by stroke aggregation, which generates the chips, i.e. the text strings. Finally, chip filtering and chip extension are employed to filter out false positives and find false negatives.

Jung et al. [17] also make use of texture in their system. First, textural properties of the RGB color bands are extracted. Then, a neural network is employed to train a set of texture discrimination masks for text and non-text classes. The inputs of the classifier are the color values of the neighborhood of a given pixel value. Chen et al. [8] use a number of weak classifiers upon which they base their decisions. To determine which image features are reliable indicators of text, statistical analysis of manually labeled text regions in city images is performed. Weak classifiers are obtained by using joint probabilities for feature responses on and off text. These weak classifiers are then used as input to an AdaBoost machine learning algorithm to train a strong classifier [41]. An adaptive binarization and extension algorithm is applied to those regions selected by the cascade classifier. The overall algorithm has a success rate of over 90% (evaluated by complete detection and reading of the text by commercial OCR software) on their test set.

Interested readers may find more information on some of the presented systems in the survey conducted by Jung et al. [18], and the one by Liang et al. [24].

### 1.5.3 Text Detection and Reading in Videos

While text detection and recognition in videos poses many of the same challenges as in images, there are a number of differences. Particularly, many text detection systems for videos assume that text is a caption overlaid on the video stream. Some general assumptions for caption text are that all characters are of a single color, that they show a strong contrast to the background, and that they are not distorted by rotation or perspective. Furthermore, it is assumed that the background is in motion while the text is not. All of these assumptions cannot be applied to text in natural scenery, and hence we will not provide an in-depth examination of work related to this area. However, some approaches are presented here that may also be of interest for text detection in images.

For instance, the video text detection system presented by Smith and Kanade [39] closely resembles the techniques used in some of the systems above. Text regions are detected by applying a  $3 \times 3$  horizontal differential filter over the input image, followed by thresholding to obtain vertical edges. A smoothing step filters out small edges. Then, adjacent edges are connected, and a bounding box computed. A set of heuristics, such as aspect ratio, fill factor, and size are used to filter out false positives.

Shim et al. [36] make use of the gray-value intensity homogeneity of text regions. Pixels with similar gray levels are merged into groups. After removing large regions as background, text regions are sharpened by a region boundary analysis. Finally, candidate regions are filtered using heuristics. While the assumption of a homogenous gray level for characters may not hold for general text in natural scenery, it is often the case for text found on signs.

Chun et al. [9] use a combination of Fourier features and a neural network classifier to detect text regions. A related approach is taken by Li et al. [23], but does not incorporate any explicit feature extraction stage. Instead, a neural network combines the outputs of the three color bands into a single decision about the presence of text. Boxes are then generated based on the neural net output. Whether or not such a classifier can be extended directly to text in natural scenery remains questionable.

Finally, Sato and Kanade [35] present a complete text reading system for videos. A simple OCR system is employed that classifies characters by grayscale correlation. The proposed OCR engine is used in combination with the text detection method by Smith and Kanade [39].

## 2 System Overview

When designing a text reading framework, it was crucial that it was flexible, simple and could incorporate a variety of text detection and recognition techniques. Overall, the main requirements for such a framework are:

**Flexibility:** Various text detection and reading techniques should be carried out in the system framework. A fixed module pipeline should not be assumed, as normally found in OCR systems. Rather, a flexible combination of recognition techniques should be allowed.

**Simplicity:** The framework should be kept small and simple. The focus should not lie on the evaluation of a system framework, but on the recognition techniques used in conjunction with it. The purpose of the framework should be to bind these techniques together.

**Traceability:** In performance measurement, it is crucial that error can be assigned to a specific module, rather than evolve out of an unknown combination of processes. It should be possible to “open” the pipeline at any stage and perform a component based evaluation.

**Completeness:** The system should be able to fulfill both the tasks of text detection and text reading, from start to finish. Text detection may or may not be a building block to the entire text reading process.

In the following we will give an overview of the text recognition framework used in this work. We will begin by giving a high-level overview over the entire system, followed by the descriptions of each recognition phase.

As shown in Figure 4, the text recognition system consists of two main steps. The first deals with generating character hypotheses, while the second groups the set of generated character candidates to words. Note that an explicit binarization or connected component extraction step may be involved, but is not necessarily, as in most traditional OCR systems. The reasons for this are two-fold. On the one hand, this general approach satisfies our requirement of flexibility, i.e. we do not want to incorporate fixed modules into the processing pipeline. Secondly, although an image preprocessing step is performed, we would like to refrain of any fixed image simplification tasks before-hand, as each of these simplifications involves a loss of information, and usually opens up the possibility of error (as in the binarization case). Rather, image preprocessing is done on a “pull”-basis, only performed if a certain module requires it. Once an image is preprocessed, features are extracted and used for generating character location candidates. These may specify merely the presence of characters, or the characters themselves. Finally, the grouping model groups character candidates to words.

### 2.1 Character Hypothesis Generation

Character candidates are generated in a number of steps. First, the image is preprocessed in preparation for the feature extraction step. The choice of feature extraction method determines the preprocessing operations required. For instance, some feature extraction methods work on the edges of the input image, while others work on the thinned components. Thus, preprocessing is done ad-hoc, whenever feature extraction requires it. The extracted features are used as evidence for generating character candidates. The framework allows to generate candidates with varying degrees of specificity. For instance, some features may merely indicate the presence of a character at a certain position, while others may indicate the actual character glyph. The first problem shall

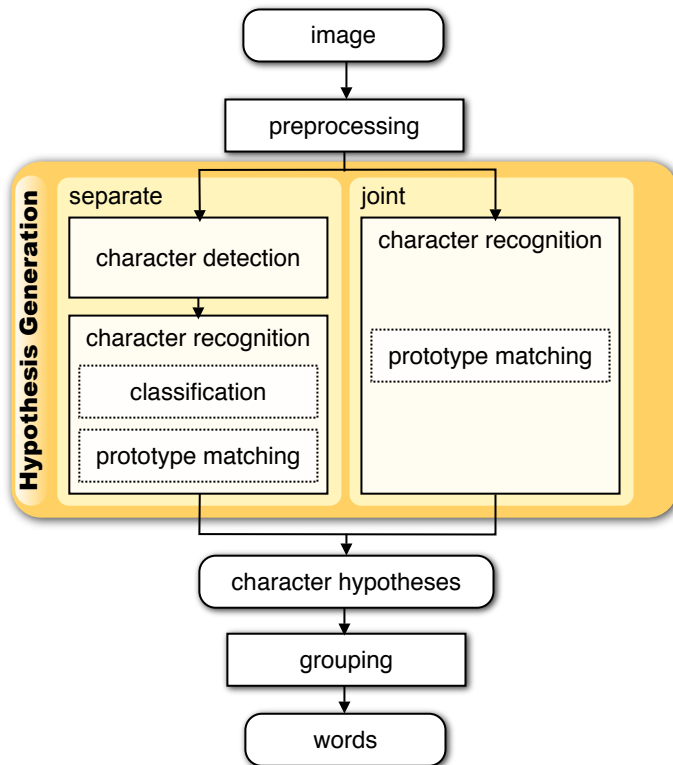


Figure 4: Steps in the text recognition system.

be referred to as *character detection*, and deals with generating a set of character *bounding boxes*  $B = \{b_1, \dots, b_n\}$ , where a box  $b_i$  is specified by the 4-tuple:

$$b_i = (x_i, y_i, w_i, h_i)$$

and  $x_i, y_i$  specify the center coordinates (in pixels) of the character within the image, and  $w_i, h_i$  specify the width and height (in pixels) of the  $i$ 'th character.

The second task deals with the actual character classification. It enriches the bounding box parameters with a glyph specification  $g$ . This task is called *character recognition* in the following. The output of this step is a set of *character hypotheses*  $C = \{c_1, \dots, c_m\}$ , each containing the character glyph  $g_i$  and its bounding box  $b_i$ :

$$c_i = (b_i, g_i).$$

Figure 4 (top) also shows the two general approaches used in this work to obtain character hypotheses. The pipeline on the left consists of *separate character detection and recognition* steps. Here, image features are used to generate a set of bounding boxes, which are then passed on to the character recognition module. The recognition module then generates character hypotheses from the image portions specified by the bounding boxes. In short, this approach is a divide-and-conquer method, where we divide the problem of recognizing characters in random imagery to the sub-problems of segmenting a random image into character regions, and recognizing a single character at a known position and scale within this region. This approach is taken by the majority of text reading systems. While this division of a problem into subproblems simplifies the entire challenge, it does come with a few caveats. Most notably, as with all operation sequences, each

component in the sequence may introduce error. That is, errors made in the detection phase will propagate through to the recognition phase. These errors include missed character locations, false positives, or incorrect bounding boxes generated by linked or split characters in the image.

Thus, we introduce the approach of *joint character detection and recognition*, shown in Figure 4 (top) by the pipeline on the right. As the name suggests, this approach does not distinguish between a character detection and recognition phase. Instead, image features are used to generate character hypotheses directly. While this approach is more complex and usually computationally more expensive, the fact that it does not include a separate segmentation step eliminates problems, such as linked and split characters.

Furthermore, the character recognition step can be divided into the following two techniques:

- **Feature classification:** The image portion is represented in a (more compact) set of features. These features should show little variance within the same character class, and a high variance for differing character classes. The features are then classified to character hypotheses, given some classification model, such as a decision tree or nearest neighbor model.
- **Image prototype matching:** A number of labeled character prototypes are fitted to the bounding box. The label of the best matching one, given some matching measure, is selected as the character class.

While both techniques can be used when the character detection and recognition steps are separated, the joint approach is based on prototype matching alone. The reason for this is that it is difficult to extract compact features from an entire image that would give rise to all character locations. This would require a classifier capable of reconstructing the original character locations based on the set of features only.

In this work, character hypothesis generation methods using both a separate and joint detection and recognition are discussed, to address questions such as how the number of processing steps (each introducing error) affects the overall result and how computationally expensive each approach is. A number of character detection and classification techniques are discussed and evaluated, some common to the OCR domain, some from related pattern recognition domains. The performance of these techniques is detailed in the performance evaluation in Section 5.

## 2.2 Filtering

The hypothesis generation techniques discussed so far have always enriched given hypotheses with new data: the character detection phase enriches the simple hypothesis of characters being within the image (an assumption we always make) to the actual positions of the characters within the image. The character recognition phase enriches given bounding box hypotheses with character classes. Filtering, on the other hand, generates a subset of hypotheses  $H'$  from a set of given hypotheses  $H$ , so that  $H' \subset H$ . Filtering hypotheses is important as initially the aim is to rather produce too many hypotheses, than too few (or in information retrieval terms: A high recall is prioritized over a high precision). The reasoning behind this is that it is generally easier to filter out false hypotheses than to subsequently generate the missed ones. To filter out characters, analysis methods are required to determine whether or not a certain hypothesis is likely to be correct or not. This is usually done by employing heuristics. Filtering is discussed in more detail in Section 3.6.

## 2.3 Grouping

Once the character hypotheses have been determined, they are grouped to words in the grouping step. In this step, two tasks are performed. Firstly, character candidates are assigned to groups, based on a number of constraints. Secondly, character locations are rejected that do not fit into any group. Furthermore, the grouping module must be able to cope with the following challenges:

- Deal with clutter: False character locations may be distributed among the true character hypotheses. The grouping module must be able to distinguish clutter from true characters.
- Deal with multiple overlapping character boxes: There may be multiple character hypotheses for the same glyph in the image. In these cases, the grouping module must make a decision on which characters to accept and which ones to reject.

The grouping of characters to words allows the evaluation of recognition performance on real-world data. In this case, performance is tested on text found in photographic images. This also allows a comparison of the performance of techniques used in this work, to those of existing systems. While much more elaborate grouping methods exist, this work will not go beyond a comparatively simple method here to keep the focus on character classification methods. Furthermore, higher level grouping techniques are not included, such as grouping to sentences or layouts as this would be beyond the scope of this work.

Finally, it should be noted that the output of the grouping module can be seen as evidence for new character hypotheses. For instance, if a group is found containing the word 'amburger', a new location could be generated to the left of the word in hopes of finding the missing 'h'. While providing interesting possibilities, grouping will be used as a final step here only.



	<b>separate approach</b>	<b>joint approach</b>
<b>preprocessing</b>	edge detection (Sobel, LoG, Canny)	
	distance transform	
	skeletonization	
<b>character detection</b>	closed contours	
	stroke detection	
<b>character recognition by feature classification</b>	orientation histograms	
	Fourier descriptors	
	skeleton features	
<b>character recognition by prototype matching</b>	template matching	
	RAST	

Table 1: Overview of the techniques used in preprocessing, character detection and character recognition. The joint approach requires no detection step and uses prototype matching techniques.

### 3 Character Hypothesis Generation

In this section an overview of the character hypothesis generation is given. Specifically, the types of features used in this work are detailed, how they are extracted from the input image, and how they can be used to generate character hypotheses. Table 1 gives an overview of the individual techniques that are used in this work for the preprocessing, character detection and character recognition steps. The table also shows that the joint approach of recognition requires no character detection step and uses prototype matching for character recognition. This section is structured according to the table: The first section deals with the processing steps involved in the separate and joint approaches. Next, the individual image operations and feature extraction techniques are discussed, starting with preprocessing, followed by an examination of the features used for character detection. Finally, the character recognition methods by feature classification and by prototype matching are discussed.

#### 3.1 Approaches

Before the character detection and recognition techniques are discussed, an outline of their use in the separate and joint approaches is given here. These two approaches can be seen as the high-level framework, into which the individual preprocessing, feature extraction, classification and matching methods can be inserted.

##### 3.1.1 Separate Detection and Recognition

In the separate detection and recognition approach a character detection step is performed before recognition resulting in a set of bounding boxes  $B$ . It is assumed that each of these boxes contains exactly one character. Then, there are two ways to classify the character within the box: recognition by feature classification and recognition by prototype matching. The classification approach segments a character region into distinct components and extracts features from the component that represents the character. That is, given a set of components  $C_b$  in a bounding box  $b \in B$ , find

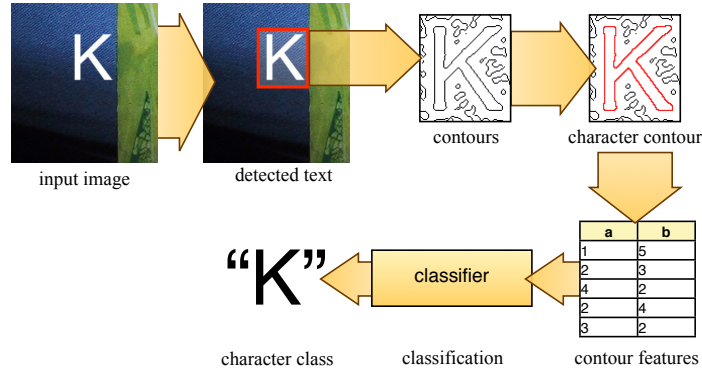


Figure 5: Steps involved in the separate approach using character recognition by feature classification. Here, a contour based classifier is shown.

the component  $c \in C_b$ , that describes the character. In this work, components are either contours or skeletons, and the character component is the outer contour or skeleton that belongs to the character glyph. There are many possibilities of extracting the character component from a bounding box, such as filtering the components using character heuristics. Here, for each bounding box the largest component that fits into it is taken, so that each box results in at most one component. This approach also shows the necessity of a character detection step, as passing the entire image to the recognition engine would result in a single character being processed only. Thus, separate detection and recognition can be broken down into the following steps:

- Run text detection to find the set of character bounding boxes  $B$ .
- For each box  $b \in B$ :
  - Find the largest component  $c$  in  $b$
  - Extract features from  $c$
  - Classify features
  - Return character class

Figure 5 illustrates this approach for the case of feature classification based on contours. Note that there are many ways to classify the extracted features to a character class. A discussion of the classifiers used in this work is given in Section 3.5.4.

Besides feature classification, prototype matching techniques (discussed in Section 3.6) can be used as a recognition method on bounding boxes as well. In Figure 5, a prototype matcher would replace the character contour identification, the feature extraction and the feature classification steps. Instead, multiple character prototypes would be matched to the image components within the detected bounding box, and the character label of the best match chosen. With the additional bounding box information, such techniques can be optimized by making the following assumptions:

- There is only one character to match within the bounding box.
- The character is roughly at the center of the box.
- The character’s width and height are roughly identical to the box’s dimensions.

While these assumptions may greatly simplify the prototype matching methods used in this work, they do not necessarily hold when dealing with distorted or rotated characters. Section 5 deals with these issues in more detail in the performance evaluation of prototype matching on bounding boxes.

### 3.1.2 Joint Detection and Recognition

While the approach of recognition by feature classification provides a simplification to the text reading task by dividing it into two sub-tasks, it does come with a few drawbacks. Firstly, even if a highly accurate recognition method is used, it will fail if the correct bounding boxes were not found in the detection step. Furthermore, the segmentation of the features into character components is not robust against linked or broken characters.

Prototype matching, on the other hand, works more closely on the image data, and requires no segmentation at all. Rather, the image is seen as a 2-dimensional signal of fixed length, in which a smaller template signal must be matched. In order to use prototype matching for character detection and recognition, a set of character prototypes is employed, and each is matched to the image. In each such run, the prototype may match multiple locations, producing high matching scores at each such location. A threshold may be used to limit the number of matching locations. The bounding boxes and prototype labels of the matches that exceed this threshold are then returned as the set of character hypotheses.

Since prototype matching requires no segmentation, the matching algorithms presented here are robust against linked or broken characters. Even occluded characters can still be matched if enough image information is available to provide a good match with a template. Without bounding box information however, prototype matchers must conduct an extensive search for characters with various fonts, scale and possibly rotation and distortion. Searching over all possible combinations is not computationally feasible. For this reason, naive solutions like template matching will only be of limited use in the joint recognition approach. The RAST geometric matching algorithm discussed in Section 3.5.2 on the other hand, promises to find character candidates throughout the image in a reasonable amount of time.

## 3.2 Preprocessing

During preprocessing, the query image is prepared for character hypothesis generation. The goal of this step is to make those image features apparent that are specific to the feature extractors or matching algorithms. For instance, the edges found during edge detection provide the input to contour based hypothesis generation. In the following, a number of preprocessing operators are presented. As many of the subsequent operations are based on edges, the edge detection methods are discussed first.

### 3.2.1 Sobel Operator

The Sobel operator [40] is an isotropic discrete two-dimensional differentiation operator typically used for edge detection. It computes an approximation of the gradient of the image intensity function. The output of the operator is a gradient map, where each pixel is either the corresponding gradient magnitude or its direction.

The operator uses two  $3 \times 3$  kernels, one for the horizontal direction and one for the vertical, which are convolved with the original image to calculate the approximation of the derivation. For a given



Figure 6: Sample image (*left*) with Sobel edge detector magnitude (*middle*) and direction (*right*) output.

input image  $I$ , the horizontal and vertical gradients  $G_x$ , and  $G_y$  are calculated as follows:

$$G_x = \frac{1}{8} \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * I \quad \text{and} \quad G_y = \frac{1}{8} \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I.$$

Then the gradients can be combined to obtain the magnitude

$$G = \sqrt{G_x^2 + G_y^2},$$

and the gradient's direction by

$$\phi = \arctan\left(\frac{G_y}{G_x}\right).$$

In this work, both the Sobel magnitude and gradient are used in the character recognition process. However, as the Sobel operator does not output thin edges, i.e. edges that are only 1 pixel thick, other edge detection methods are used when thin edges are required. Figure 6 shows an example image and the corresponding gradient direction and magnitude maps obtained by the Sobel operator on a gray-level version of the input image.

### 3.2.2 Laplacian of Gaussian

The *Laplacian* operator  $\Delta$  is an isotropic operator which approximates the second spatial derivative of an image. Coupled with a *zero-crossing detector*, the Laplacian can be used for edge detection, as the zero points mark the local maxima of the first derivative (points of rapid intensity change). In order to reduce its sensitivity to noise, the image is often first smoothed using a Gaussian operator, given by

$$G_\sigma(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{x^2 + y^2}{2\sigma^2}\right].$$

This results in the *Laplacian of Gaussian* (*LoG*) operator, given by

$$\begin{aligned} LoG * I &= \Delta[G_\sigma(x, y) * I(x, y)] \\ &= [\Delta G_\sigma(x, y)] * I(x, y). \end{aligned}$$

Note that use of the associativity of the convolution operator is made, which allows for the following optimization: Instead of convolving the image twice, the Gaussian is applied to the Laplacian mask, and the obtained Laplacian of Gaussian mask is applied to the image. For instance, the LoG can

be approximated by the following  $5 \times 5$  kernel:

$$LoG = \frac{1}{16} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} * I$$

To find the edge points, a zero-crossing detector marks those pixels where the sign of the Laplacian changes, with the strength of the crossing. Here we define the strength as the minimum absolute value of the two values involved in the sign change. Finally, this map is thresholded to obtain a binary image. Figure 7 shows the filter applied to a test image with various settings for the Gaussian standard deviation  $\sigma$  and the crossing threshold  $t$ .

There are 2 notable advantages of the LoG operator in comparison to other edge detectors. Firstly, it produces thin edges (i.e. edges of 1 pixel thickness), as only the crossings through zero are marked. Secondly, it produces closed contours, or contours that are bounded by the image border, that is, it segments the image into blobs. On the downside, the LoG operator is very sensitive to noise for low values of  $\sigma$ . Furthermore, as Figure 7 shows, if the threshold  $t$  is set too high, the second property does not hold. Thus,  $t$  introduces a trade-off between cleaner segmentation and the amount of noise.

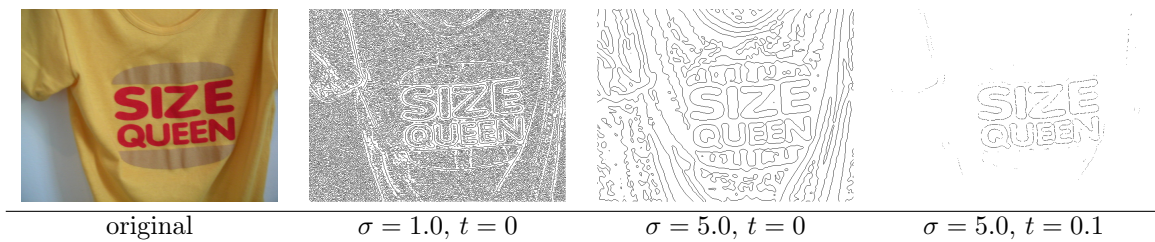


Figure 7: Sample image (*left*) and Laplacian of Gaussian results for various settings of  $\sigma$  and  $t$ .

### 3.2.3 Canny Edge Detection

The *Canny* edge detector [6] also produces thin contours, but obtains these in a different manner. The steps are outlined in the following:

1. Noise reduction: The image is convolved with a Gaussian mask to reduce noise.
2. Intensity gradient extraction: Edges in the image are classified by their direction angle  $G$ . Many edge detectors, such as the Sobel operator (discussed above), return the strength of the horizontal and vertical edge, from which  $G$  can be computed. Usually, the direction angle is rounded to one of four angles, representing horizontal, vertical, and the two diagonals.
3. Non-maximum suppression: Given the rounded edge gradients, a search is performed to determine if the gradient magnitude is a local maximum orthogonal to the gradient direction. This is done by comparing the pixel in question with its neighbors that do not lie in the direction of the gradient. For instance, if a pixel lies on a horizontal edge, the edge intensities directly above and below the pixel in question are considered. If these intensities are lower than the pixel's intensity, it is marked as an edge.

4. Tracing: The edges are traced using thresholding with *hysteresis*, i.e. two thresholds ( $T_{high}$  and  $T_{low}$ ) are required for tracing. A trace is initiated only if the intensity is greater or equal to  $T_{high}$ . The edge is then traced as long as the intensity does not drop below  $T_{low}$ . This allows tracing even faint edges, as long as there is a strong starting point to begin with.

The Canny edge detector was designed to be optimal in the following ways:

**Good Detection:** The algorithm should be capable of finding as many real edges in the image as possible.

**Good Localization:** The edges marked should be as close to the original edges as possible.

**Minimal Response:** The edges should be thin, i.e. each edge should be marked only once.

Typical parameters for the Canny edge detector are the Gaussian smoothing factor  $\sigma$ , and the hysteresis thresholds  $t_{high}$  and  $t_{low}$ . In the implementation used here, rather than setting these to constant values, we employ an image adaptive approach: Let  $P_E$  denote the distribution of edge magnitudes. Furthermore, let  $q_N$  equal the  $N$ -quantile of  $P_E$  for some selected  $N$ . Then, given two factors  $a_{hi}$  and  $a_{lo}$ , both greater than or equal to 1.0, we calculate the thresholds by

$$\begin{aligned} t_{hi} &= a_{hi} \cdot q_N, \\ t_{lo} &= a_{lo} \cdot q_N \end{aligned}$$

In other words, the selection of  $N$  gives the amount of noise we expect in the image. The thresholds are then set to some multiple of the quantile, ignoring all edges in the noise range. Figure 8 shows an example image and the corresponding Canny edge maps for various parameter values.

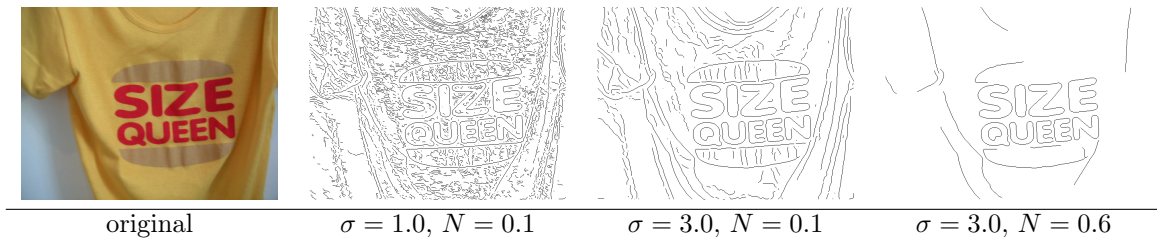


Figure 8: Sample image (*left*) and the Canny edge detection output for various values of  $\sigma$  and  $N$ . The hysteresis factors were set to  $a_{lo} = 2.0$  and  $a_{hi} = 4.0$ .

### 3.2.4 Distance Transform

Applying the distance transform [2] on a binary image yields a map that supplies each pixel with the distance to the nearest boundary pixel. More formally, given a set of  $n$  boundary pixels  $B \in \{(x, y)\}^n$ , the distance transform computes a map  $D$  for the distance metric  $m$ , where

$$D_{u,v} = \min_{(x,y) \in B} \|(x, y) - (u, v)\|_m$$

A transform is qualified by the distance metric it uses. Common candidates are the Euclidean ( $m = 2$ ), Manhattan ( $m = 1$ ), chessboard ( $m = \infty$ ), or Chamfer metrics. Figure 9 shows the distance transformation performed on a binary image using various distance metrics. In the original image, the black pixels represent the non-boundary pixels.

In this work, the algorithm described in [28] was implemented, which allows the computation of the Euclidean, Manhattan and chessboard distance transform in two passes. The acquired distance maps can then be used for measuring the width of components, which is used for stroke analysis discussed in Section 3.3.2.

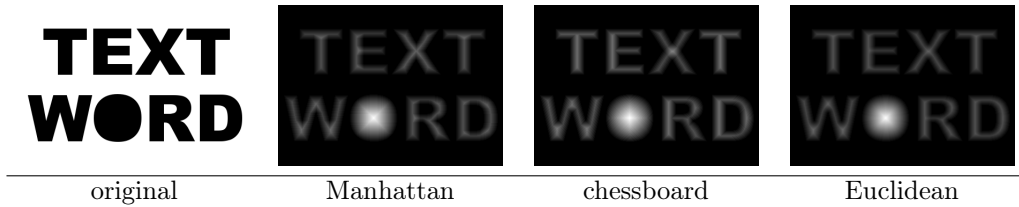


Figure 9: Sample binary image (left) and the corresponding distance transform using the Manhattan, chessboard, and Euclidean distance metrics.

### 3.2.5 Connected Components

Generally, it is preferred to avoid any type of segmentation in the preprocessing phase. However, the connected component step described here is not meant for segmenting characters from clutter, but used to identify the set of *all* connected regions in the input image. Later, these components are used for skeleton extraction as described in the next section.

The extraction of the connected components from a color or gray-scale image can be described as follows: In a first step, edge detection is applied to the input image. The resulting edge map is thresholded to obtain a set of boundary pixels. The goal is to group all those pixels that are not completely separated by a boundary. This is done by *labeling* the image, i.e. every pixel is assigned an ID that is unique to the component it belongs to. The labeling algorithm requires two passes over the image. The first pass labels each pixel with the label of its non-boundary neighbors. If a pixel does not have a non-boundary neighbor, that has been labeled already, it is assigned a new label. If a pixel has neighbors of conflicting labels, these are stored as equivalent. In the second pass, each pixel is relabeled to the lowest equivalent label. Figure 10 shows an input image, the binarized edges, and a visualization of the assigned label IDs.

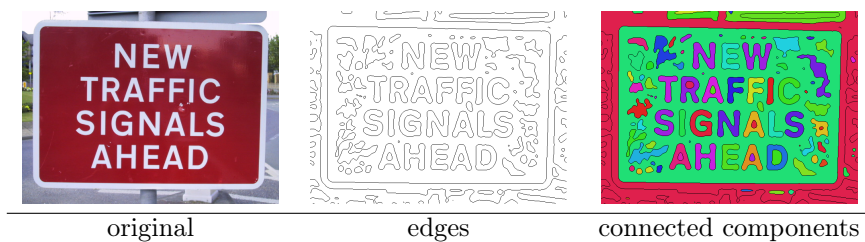


Figure 10: An example input image (left), the binarized edges (middle), and the extracted labels (right), visualized here by assigning a random color to each connected component.

### 3.2.6 Skeletonization and Thinning

A topological skeleton of a shape is a thin version of that shape, where every point is equidistant to at least two boundary points. The skeleton should largely preserve the extent and connectivity of the original region. There are a number of methods to obtain the skeleton from a binary



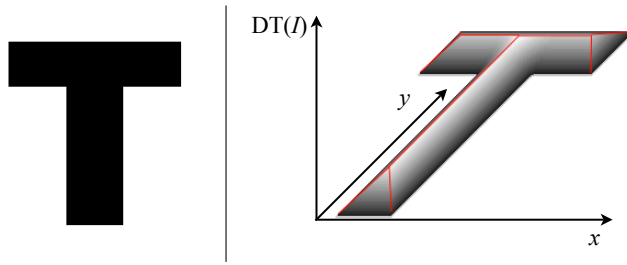


Figure 11: The distance map of the object on the left extruded into 3D-space (*right*). The ridge points are the singularities of the distance function, marked in red.

image. The first is to use morphological thinning, which successively erodes away pixels from the boundary. Obviously, for a binary image and its distance map  $D$ , this process will require  $\max[D]$  passes over the image. For large characters often found within high-resolution photographs, this leads to a computationally expensive operation. An alternative approach is to first calculate the distance transform of the input image. The skeleton then lies along the singularities of the distance transform. Figure 11 shows a distance transform projected into 3-dimensional space. The singularities are highlighted in red.

The skeletonization algorithm must be carefully selected, as apart from efficiency many algorithms lead to very different results. Small changes or noise in the shape form usually lead to vastly different skeletons. This problem is amplified by the discrete property of the contours we deal with. Thus, it is desirable to obtain skeletons that are robust to small changes, even if the skeletons are not complete. Some algorithms, such as the morphological ones, are able to produce fairly clean thinned versions of the original shape, that are robust against noise. Other algorithms require a separate pruning step, in which unwanted skeletal branches are removed. Often, as in our case, branches reaching out to the boundary are unnecessary and can be removed. Figure 12 shows a sample character, and its associated raw and “cleaned” skeletons.

In this work, both a morphological, and a distance-map based skeletonization algorithm were implemented. The morphological algorithm uses standard thinning masks, such as the ones described in [12]. For the distance-map based approach, a number of algorithms exist to efficiently find the singularities in a distance map. Here, a modified version of the algorithm found in [7] is used.

Additionally, an extensive pruning step is performed to greatly simplify the skeleton form. For this, we make use of an important property of text: Glyphs are often a composition of *strokes*, that is, they are composed of shapes that have roughly parallel contour lines (such as the character shown in Figure 12 (left)). Thus, a point is considered as a skeletal point only if the normals connecting it to the shape outline are roughly pointed in opposite directions. This will for instance ignore all those spurious branches reaching out to shape corners, such as those marked red in Figure 12 (middle). To find the direction of the normal for a skeletal pixel  $(x, y)$ , the distance map is used again. Use can be made of the property that the distance values on the normal drop the fastest in the direction of the contour. That is, for a set of angles  $R_i$ , the normal angle  $\varphi_i \in R_i$  at  $(x, y)$  is approximated by

$$\varphi_i = \operatorname{argmin}_{\varphi \in R_i} [D(x - \alpha \cos \varphi, y - \alpha \sin \varphi)].$$

The angles  $\varphi_1$  and  $\varphi_2$  of both normals are obtained by including all angle candidates in  $R_1$ , but only those roughly opposite to  $\varphi_1$  in  $R_2$ . The point  $(x, y)$  is then marked as a skeleton point only if the difference of angles  $\delta(\varphi_1, \varphi_2)$  lies within  $180^\circ - \epsilon < \delta < 180^\circ + \epsilon$ , for some tolerance value  $\epsilon$ .

Together with other image features, such as color or the distance map, skeletons can be enriched with additional information. For instance, mapping the points of a skeleton to the distance map, yields the stroke thickness of the skeleton at those points.



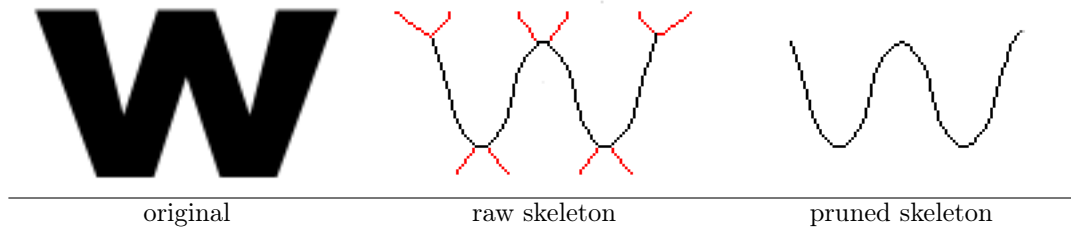


Figure 12: A sample character (*left*), the corresponding raw skeleton (*middle*), and the pruned version (*right*).

### 3.3 Character Detection

In the following the task of character detection is examined. Given the image descriptors obtained in the preprocessing step, the goal is to find features that will act as evidence for determining the bounding boxes of characters.

#### 3.3.1 Closed Contours

As characters are expected to contrast highly with their backgrounds (in order to be readable), it may be assumed that their edges form closed contours that fully separate the character pixels from the background pixels. Thus, all chains of edges which form a closed contour may be considered evidence for a character at the location of this contour. An edge-chain with two end-points is considered closed, if the distance of the two endpoints does not exceed a certain threshold  $d_c$ . If an edge chain has more than two end-points, it is split into smaller chains, until each chain has exactly 2 end-points. Setting  $t_c$  to anything greater than 0 allows identifying even those contours with a spurious opening in them.

However, as we are often confronted with broken contours that have multiple small openings in the trace, we apply a *contour joining* preprocessing step: All end-points that are sufficiently close to one another are connected, and *joined* to a single contour. The threshold of the joining distance must be selected carefully: If selected too low, small amounts of noise may become connected to form a large contour. On the other hand, if selected too high, large or high resolution character contours may be left unconnected. For this reason, the threshold is chosen adaptively to the contour length. Specifically, two contour end-points  $p_1$  of contour  $C_1$ , and  $p_2$  of another (or possibly the same) contour  $C_2$  are joined iff

$$\|p_1 - p_2\| \leq d \cdot \max[|C_1|, |C_2|],$$

where  $|C_i|$  is the length of contour  $C_i$ , and  $d$  is some real-valued weight, usually in the range  $[0, 1]$ . The contour joining algorithm performs a joining step, in which all end-points are analyzed and joined if they satisfy the joining constraint. As this may result in new larger contours, the process is repeated until no more joins have been performed. While the complexity of each iteration has a worst case of  $O(n^2)$  for  $n$  contours, a scan-line implementation, in which the points are processed from top to bottom, performs much faster in most of the cases.

#### 3.3.2 Strokes

The stroke map  $S$  of the input image is a binary map, in which each value  $S(x, y)$  indicates whether  $(x, y)$  lies on the central axis of a stroke. These have been extracted by either tracing ridges along

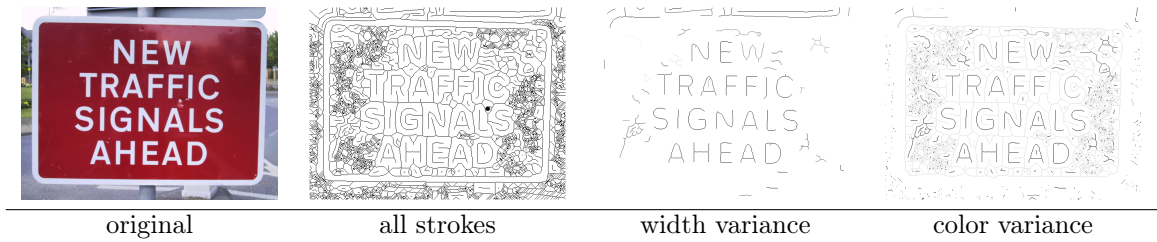


Figure 13: The stroke width and color variance for an example image (*left*). The brightness of each stroke indicates its variance, where black is a low variance and white a high variance.

the distance transform, or thinning the connected components of the input image. Note that the former can be treated as evidence for character locations already, as the ridges extracted through tracing have been pruned to include only those points that lie within parallel contour boundaries. As this is a typical feature of text, areas containing these strokes are also more likely to contain text.

However, combining the binary stroke maps with other maps obtained in the preprocessing step can further help distinguish between regions of text and those of background. An analysis of a large number of text strings in natural scenery suggested that the stroke width of text found in typical photography normally does not vary much within the character glyph. The use of stroke width to segment text from background has also been used in [25]. Here, use is made of this observation by extracting the *stroke width variance* of a skeleton, which describes how much the width within a shape varies. Another common property of text found in photography is that color often remains largely unchanged within each character. Again, the variation of color can be used as an indicator for text likelihood.

More formally, if a skeleton  $\Psi$  is given by a set of points  $\Psi = \{(x_1, y_1), \dots, (x_n, y_n)\}$ , the stroke property variance  $\text{Var}_M$  of some property  $M$  is given by

$$\text{Var}_M[S] = \frac{1}{n} \sum_{i=1}^n (M(x_i, y_i) - \mu_M[S])^2,$$

where the stroke mean  $\mu_M$  is given by

$$\mu_M[S] = \frac{1}{n} \sum_{i=1}^n M(x_i, y_i).$$

Then the stroke width variance is given by  $\text{Var}_D$ , where  $D$  is the map obtained from the distance transform. Similarly, the *stroke color variance* for a color image  $I$  is given by  $\text{Var}_I$ . Figure 13 shows two sets of skeletons obtained from an input photograph with their stroke width and color variance visualized. A brighter shade of gray indicates higher variance.

### 3.4 Character Recognition by Feature Classification

The previous section dealt with finding character candidate bounding boxes only. Now, the task of recognizing characters found within candidate regions to text is addressed. This section deals with recognition methods, which classify characters based on the features extracted from the character contour or skeleton. This is called the *feature classification* approach. In the following feature extraction methods are introduced based on contour orientation, fourier descriptors, and skeletal features. The final part of this section deals with the classification of the extracted features.

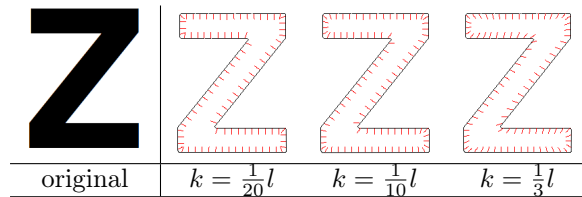


Figure 14: Original shape (*left*), and the contour normals (*right*) shown in red for various values of the smoothing parameter  $k$ , and  $\epsilon = 1.0$ . The total contour length is given by  $l$ .

### 3.4.1 Orientation Histograms

The closed contour of a character is a piecewise linear curve that separates the character from the background. While the parametrization of the contour itself may be used as a feature vector, without normalization this would not yield any robustness against translation, scale, rotation, or starting point. Much research has been conducted on how to capture contour information in a simple, usually one-dimensional and discrete function. For instance, by taking all the left-most and all right-most contour points from top to bottom, two 1-dimensional functions are obtained, called *profiles* [38]. While this approach simplifies a set of 2D feature values to 1D, it does not provide rotation invariance (translation and scale invariance are simple to obtain). In another approach, the contour is represented as a list of chain codes, where each such code gives the offset to the next pixel. That is, in an  $n$ -connected neighborhood, there are  $n$  distinct codes. While these codes can be made somewhat robust against rotation and starting point, this invariance only holds for a finite number (namely  $n$ ) degrees of rotation, and only if the contour is robust against any of these transformations [12]. Kimura and Shridhar [19] obtain starting-point invariance by taking the distribution of chain code values over the contour. Here, this approach is extended to histograms of orientations. Specifically, instead of representing the contour as a set of discrete connected points, a continuous representation is used, in which the contour is given by the following two dimensional function

$$z(p) = (x(p), y(p))^T,$$

which yields the contour point at the path length  $p \in \mathbb{R}$  when traversing the contour clockwise. Then, the orientation at  $p$  is given by

$$\alpha(p) = \arctan \left( \frac{y(p + \epsilon) - y(p)}{x(p + \epsilon) - x(p)} \right),$$

for some small real-valued  $\epsilon$ . A smoother set of angles can be obtained by calculating a range of angles  $[\alpha(p - k\epsilon), \dots, \alpha(p), \dots, \alpha(p + k\epsilon)]$ , for some discrete value  $k$ . Figure 14 shows the orientations (as normals) sampled at a discrete number of points for various settings of  $k$ . Note, how a higher value of  $k$  tends to produce a smoother set of angles, so that an angle close to a corner is influenced by the change in direction. The orientations can be made invariant to the starting point by considering the frequencies of orientations only. The normalized frequencies of such an orientation histogram can then be used as features for classification.

Finally, this approach also allows the normalization of the frequencies for invariance against rotation. To do this, the *principal axes* [12] of the shape specified by the contour are calculated. This is done in the following manner: Consider the set of points that fall within the shape interior. Assuming these samples following a normal distribution, the mean vector and covariance of this distribution is calculated. According to the principal axis theorem, the eigenvectors are then the principal axes of the ellipse induced by the covariance. As these axes are fairly robust against rotation (modulo  $180^\circ$ ), the angle of the first principle axis is subtracted from all orientation angles

axes				
histogram				
axes				
histogram				

Table 2: Two rotated shapes along with their principle axes (red), and their normalized orientation histograms.

to obtain the normalized angles. Figure 2 shows several example shapes along with their principle axes, and the extracted normalized orientation histograms.

### 3.4.2 Fourier Descriptors

Given a closed contour that represents the boundary of some shape within the image, the contour curve can be described in a parametric and continuous form. For a given starting point  $(x_0, y_0)$ , the parametric curve is given by the two functions  $\mathbf{x}(p)$  and  $\mathbf{y}(p)$ , that map a real-valued  $p$  to the coordinates  $(\mathbf{x}(p), \mathbf{y}(p))$  on the contour that are at length  $p$  with respect to the starting point. We can combine these two continuous curves into one, by viewing the coordinates as complex values. We then obtain the complex function

$$\hat{z}(p) = \mathbf{x}(p) + i\mathbf{y}(p),$$

which satisfies the constraint

$$\hat{z}(p + nL) = \hat{z}(p) \quad n \in \mathbb{Z},$$

where  $L$  is the total length of the contour, i.e. the function  $\hat{z}$  is periodic. We are essentially treating the  $x$ -axis as the real axis, and the  $y$ -axis as the imaginary axis. While the interpretation of the sequence is recast, the nature of the boundary remains unchanged. A periodic function can be expanded in a *Fourier series*. If we sample the boundary at  $n$  equidistant points with distance  $d$ , we may apply the discrete Fourier transform (DFT) of  $z(k) = \hat{z}(kd)$  to obtain the complex coefficients

$$a(u) = \frac{1}{n} \sum_{k=0}^{n-1} z(k) e^{-j2\pi uk/n}.$$

These coefficients are called the *Fourier descriptors* of the boundary. The original contour can be reconstructed by applying the inverse Fourier transform, given by

$$z(k) = \sum_{u=0}^{n-1} a(u) e^{j2\pi uk/n}.$$

Suppose that instead of considering all descriptors, only the first  $m$  coefficients are used, i.e. we set the coefficients  $a(u) = 0, \forall u > m$ . Then the result is an approximation of the original contour, omitting the higher frequency components of the shape descriptors. Note that the number of points in the boundary remains unchanged. Rather, the number of terms used in reconstructing it has been reduced. Figure 28 shows an example boundary transformed into frequency space and

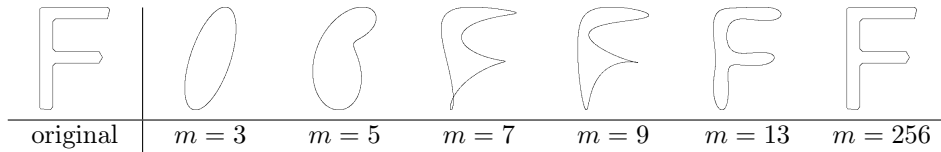


Figure 15: Contour sampled by 256 points (*left*), transformed to Fourier space, and back to the spatial domain using the first  $m$  descriptors only.

reconstructed again using different values for  $m$ . The image shows that even a small value for  $m$  already gives a good approximation of the original shape. This suggests that the first  $m$  descriptors are good feature candidates that capture the essential shape information of the contour, and that they can be used to distinguish between shapes. Without any further preprocessing however, the Fourier descriptors are sensitive to translation, scale, rotation and the starting point. We will now discuss ways of modifying the descriptors to obtain invariance against these transformations.

The first descriptor  $z_0$  expands to

$$z_0 = \frac{1}{n} \sum_{k=0}^n x(k) + \frac{i}{n} \sum_{k=0}^n y(k),$$

and gives the mean coordinate, or centroid of the boundary. Thus, to obtain translation invariance,  $z_0$  can be simply set to 0, which moves the centroid of the boundary to  $(0, 0)$ . The second coefficient describes a circle

$$z_1 = r_1 \exp(i\alpha_1).$$

If the contour is scaled by a coefficient  $s$ , all Fourier descriptors are also scaled by  $s$ . If the contour is traced counter-clockwise, the first coefficient is always unequal to zero [15]. Therefore, all descriptors can be divided by the magnitude of the second Fourier descriptor to obtain a scale invariant vector

$$z(k) = \frac{z(k)}{|z(1)|}.$$

There are various approaches of obtaining rotation and starting point invariance. The simplest of these methods is to ignore the phase, and take the magnitude of the descriptors only. However, this would result in a loss of a substantial amount of information. To visualize the importance of the phase, Figure 16 shows shape reconstructions performed with a randomly modified phase. The image shows that the contour bears little similarity to the original one. Nevertheless, the question remains of just how important phase information is for character classification. To give an answer to this question, the classification performance of the descriptor magnitudes only will be compared to a more sophisticated approach discussed next.

The second, more complex approach achieves rotation and starting point invariance while keeping the phase information intact. If a contour is rotated counter-clockwise by the angle  $\alpha$ , the Fourier descriptor  $z_k$  is multiplied by the phase factor  $\exp(ik\alpha)$ , according to the shift theorem for the



Figure 16: Contour (*left*) transformed to Fourier space and back (*right*), while randomly modifying the phase.

Fourier transform. The rotation can be normalized, by subtracting the phase shift  $k\alpha$  from all of the Fourier descriptors:

$$z(k) = z(k)e^{-j\alpha k}.$$

To achieve starting point invariance, a specific starting point is chosen in the spatial domain that is invariant to translation, scale, and (with a  $180^\circ$  ambiguity) rotation of the contour. To do this, we first calculate the principal axes of the shape specified by the contour. The starting point is then selected as the top-left most contour point that intersects the first principal axis. Figure 17 shows a number of shapes transformed to frequency space and back again, using the normalization procedures described here.

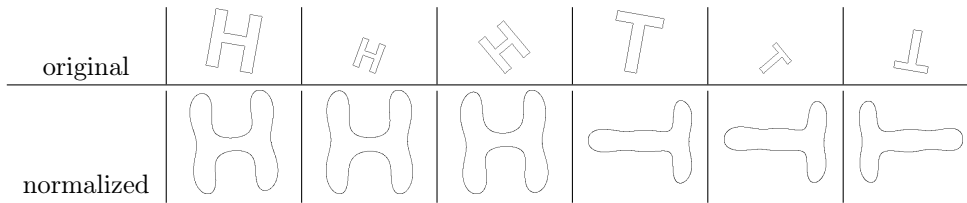


Figure 17: A number of contours (*top*) transformed to Fourier space, normalized with respect to translation, scale, and rotation, and transformed back into the spatial domain (*bottom*).

### 3.4.3 Skeleton Classification

So far, ways to classify characters have been discussed based on features obtained from character contours. In the following the focus is shifted to character classification of features taken from the skeleton. Some of the methods mentioned above, such as orientation histograms, can be applied to skeletons, requiring only slight modifications in the implementation. Fourier descriptors, on the other hand, require a defined order of points. As skeletons may contain branches, a more complex ordering of points would need to be found.

The advantage of using skeletons instead of contours is their invariance to character stroke thickness. That is, a bold character will have roughly the same representation as a thin one. The downside is that character representations become so simplified that we must expect a large amount of false positives in noisy areas of the background. On the other hand, this simplified representation of character can be used for character classification. Skeletons can be viewed as graphs, where each node is either an end-point, a junction or a change of direction. In this respect, skeletons capture the structure of the original character shape, and are very similar to how a human would describe a character (For instance, a “T” could be described as: two strokes, one horizontal and one vertical, connected by a T-junction). This leads to a set of discrete features that can be extracted from the skeleton graph [20, 34]:

- The total number of end-points.
- The total number of T-junctions.
- The total number of X-junctions.
- The total number of loops.
- The total number of horizontal and vertical lines.
- The total number of curves.


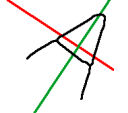
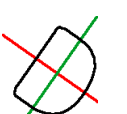
			
Total number of end-points	2	2	0
Number of end-points in zones	1,1,1,1	0,2,1,1	0,0,0,0
Number T-junctions	0	2	0
Number of X-junctions	0	0	0
Number of loops	0	1	1

Figure 18: Examples of character skeletons and their discrete feature values. The first principle axis is shown in red and the second in green.

These topological features are extracted from the graph and used for classification. Note that instead of counting curves and straight lines, which in itself is a complex task to do in a robust manner, a 4-bin orientation histogram is used, representing the amount of skeleton strokes in the horizontal and vertical directions. Additionally, the skeleton is divided into four zones, and the end-points are counted for each of these zones. Most of the features listed are robust against translation, scale and rotation. In fact, using only these features, there is no way of distinguishing an 'M' from a 'W', or a 'u' from an 'n'. For this reason, it is desirable to have a certain amount of sensitivity to rotation. To obtain invariance against rotation modulo  $180^\circ$ , the principle axes are used (see orientation histograms). Using both of these axes, the image is divided into four zones: Two zones divided by the first principle axis, and two zones divided by the second. Furthermore, the orientation of the first principle axis is used to normalize the angles in the orientation histogram. Figure 18 shows a collection of example characters, along with the zone subdivision and the discrete feature values.

### 3.4.4 Classification

Once the features have been extracted from the character component, these are classified using various well-known classification methods. A brief overview and explanation of these are given in the following:

- *k-Nearest Neighbors*: The  $k$ -nearest neighbor classifier (short:  $k$ -NN) compares a given feature vector with all feature vectors of a set of labeled training samples in a database. The  $k$  nearest neighbors are selected, and the most frequent label among them chosen. If there is a tie between two or more labels,  $k$  is decreased by 1, and the evaluation is performed again. This process will always yield a result when  $k$  reaches 1. In this work, two measures were used for the evaluation of the distance between two feature vectors  $F$  and  $G$ :

- Euclidean distance, given by  $\|F - G\| = \sqrt{\sum_i (F_i - G_i)^2}$ .
- *Jensen-Shannon Divergence* (JSD) [11], used for histogram features. The JSD is based on the *Kullback-Leibler divergence* (KLD), given by

$$D_{KL}(F, G) = \sum_i F_i \log \frac{F_i}{G_i}.$$

The KLD measures the expected difference in the information length required to code samples from  $F$  when using a code based on  $F$ , and when using a code based on  $G$ . The JSD extends the KLD to a smoothed, symmetric measure, given by

$$D_{JS}(F, G) = \frac{1}{2}D_{KL}(F, F + G) + \frac{1}{2}D_{KL}(G, F + G).$$

- *Bayesian*: Bayesian classifiers are probabilistic classifiers based on Bayes' theorem, given by  $P(C|x_1, \dots, x_n) = \frac{P(C)p(x_1, \dots, x_n|C)}{p(x_1, \dots, x_n)}$ , where  $x_1$  to  $x_n$  are  $n$  feature values, and  $C$  is the character class. The classifier chooses the class  $C$  for which the posterior probability  $P(C|x_1, \dots, x_n)$  becomes maximal. Here, we use the *naive Bayes* classifier, which assumes independence among the feature values, so that we can rewrite the above to  $P(C|x_1, \dots, x_n) = P(C) \cdot \prod_{i=1}^n p(x_i|C)$ . The  $p(x_i|C)$  can easily be estimated from the training set. Note that the denominator is removed as it is independent of the class variable. The naive Bayes classifier works remarkably well [10], despite its (often incorrect) assumption of independent features.
- *Decision Trees*: A decision tree classifier employs a tree model in which each node of the tree leads to a decision based on a certain feature value. The tree is traversed from the root, until a leaf node, representing some output class, is reached. Here, the popular *C4.5* tree is used, which sorts the decisions by information gain, and the *classification and regression tree* (CART), which incorporates a decision tree inducer for discrete classes similar to C4.5, combined with locally weighted linear regression for continuous values [42].

While the feature classifiers based on orientation histograms and Fourier descriptors use a  $k$ -NN classifier, the skeleton classification method is evaluated using all three classifiers, as it includes a mix of discrete and real-valued features.

### 3.5 Character Recognition by Prototype Matching

The methods described in this section classify characters by attempting to match labeled prototype (or model-) characters to the image. This approach is called *prototype matching*. Here, two variants of matching are discussed: The commonly known template matching, and the lesser known, but very promising RAST matching from the domain of geometric matching, which shows high robustness with respect to distortion and clutter.

#### 3.5.1 Template Matching

Template matching is a technique for finding a subregion of an image that matches a given template image. That is for an image  $I$  and a template  $T$ , the goal is to find the best matching position  $x^*, y^*$ , where

$$x^*, y^* = \operatorname{argmax}_{x,y} [M_T(I, x, y)],$$

and  $M_T(I, x, y)$  is a matching measure for the visual similarity between  $T$  and the identically sized image region of  $I$  at  $x, y$ . Using model character images as templates, this technique could be used to find characters within an image. However, on further inspection it becomes apparent that the large variation of size, font and rotation of characters within photographic images would yield a search space far too large to be of any use in simple template matching. Thus, it is assumed that the location and size of characters are known before-hand. Furthermore, the rotation of characters is ignored here.

Assuming that size and position of the characters are known, the task that remains is that of determining the best character at that position. This is done by first resizing and repositioning every character template to the target location. As templates may be resized to very large and disproportionate dimensions, the template characters are stored in a vector format. Once the templates have been fitted to the target location, a matching score is calculated. Based on these scores, the best prototype is determined using  $k$ -nearest neighbor classification, and its class label is selected as the output character class.



In this work two approaches for the definition of the matching score  $M_T$  are tested. The first operates on the image edges and calculates the correlation of the template edges of  $T$  with those of  $I$  at the target location. Thus, the algorithm does not operate on  $I$  directly, but rather on its edges, given by a binary map  $I'$ , where the edge pixels receive a value of 1, and all other pixels 0. Such a map can be obtained using the Canny or LoG (with zero crossings) edge detectors. Suppose the template edges are also given as a 2-dimensional binary map  $T_E$ , which similarly returns 1 for edge pixels, and 0 for the remaining pixels. Then, the correlation of an  $M \times N$  sized template edge map  $T_E$  in  $I$  at  $(x, y)$  is given by

$$T_E(x, y) \circ I(x, y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} T_E(m, n) \cdot I'(x + m, y + n).$$

If  $E$  is the set of edge pixels in  $T$ , i.e.  $E = \{(x, y) | T_E(x, y) = 1\}$ , the term above may be simplified to

$$T_E(x, y) \circ I(x, y) = \frac{1}{|E|} \sum_{(m, n) \in E} I'(x + m, y + n),$$

Although this method does in fact return a high correlation when template and image edges fall directly onto one another, even slight shifts in the position or scale of the template lead to a rapid decline in the correlation, quickly reaching zero. In order to be more robust against these slight transformations, the correlation is not calculated on  $I'$  directly, but on the distance transform of  $I'$ . Then, if a contour point does not fall onto an image edge directly, the score will be “penalized” by the distance to the nearest edge. As good matches then lead to low correlation values, the score is obtained by

$$M_T^1 = \exp(-T_E(x, y) \circ D_{I'}(x, y)),$$

where  $D_{I'}$  is the distance transform of  $I'$ .

In the second approach to calculating a matching score, color information in  $I$  is used directly, by calculating the color variance of the matching pixels with  $T$ . Instead of operating on template edges, consider the set of pixels that fall within the template contour. Assuming these fill-pixels are given by an  $M \times N$  binary map  $T_F(x, y) \rightarrow \{0, 1\}$ , the set of target pixel locations at  $(x, y)$  is defined as

$$\tau(x, y) = \{(x + m, y + m) | T_F(m, n) = 1\}$$

Then, the match variance  $\sigma_T^2$  at  $(x, y)$  is given by

$$\sigma_T^2(x, y; I) = \text{Var} \{I(m, n) | (m, n) \in \tau(x, y)\}.$$

Finally,  $\sigma_T^2$ , is normalized by dividing by the total color variance  $\sigma^2$  of all pixels within the target region. Thus the matching score  $M_T^2$  is obtained by

$$M_T^2 = \exp\left(-\frac{\sigma_T^2}{\sigma^2}\right).$$

Figure 19 shows the matching of two templates on a test image, and the matching scores at each point (using the edge energy method). The visualization gives an idea of how precisely the prototype and the image character must line up to obtain a high score.

### 3.5.2 RAST Matching

Another common matching technique is that of *optimal bounded error matching*, which has been used in both 2D and 3D object recognition tasks extensively [13]. These recognition tasks address

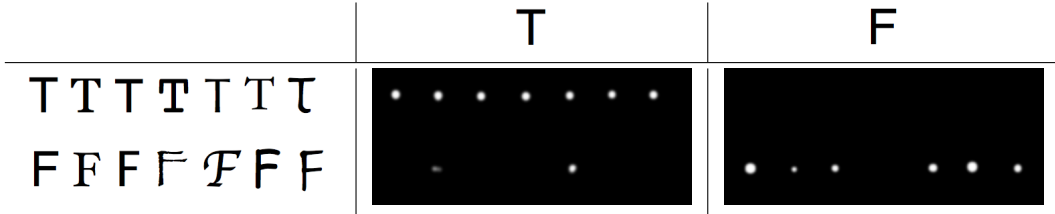


Figure 19: Two templates (*top*) matched to the test image (*left*). The matching scores are visualized for each point, where a brighter shade indicates a better match. Here, the edge-energy matching method was used.

the following problem: Assume a set of image points  $G$ , and a set of model points  $M$  are given. The goal is to find the viewing transformation (if one exists), that maps each model point to within a distance  $\epsilon$  of an image point. That is, for all transformations  $T_\theta$  over the space of parameter vectors  $\theta$ , the goal is to find a value  $\theta_0$  so that  $T_{\theta_0}$  is the “best” mapping of the model points in  $M$  onto the points in  $G$  under the given error bounds. More formally, this matching problem can be seen as the task of maximizing a match quality function, given by

$$Q(\theta; M, G) = \sum_{x \in M} \max_{y \in G} [q_\theta(x, y)],$$

and  $q_\theta(x, y)$  is a match measure for the transformation of point  $x$  onto  $y$ , using the parameters  $\theta$ . In this work, this measure is defined as

$$q_\theta(x, y) = \begin{cases} 0, & \text{if } \|T_\theta(x) - y\| > \epsilon \\ 1, & \text{otherwise} \end{cases}$$

In the 2-dimensional case a transformation is given by a translation  $t$ , a scale  $s$ , and a rotation  $r$ . Furthermore, the set of points is extended to a set of *edgels*, i.e. instead of using 2D points described by  $x$  and  $y$  coordinates alone, a third value  $\alpha$  is given which adds the notion of orientation. Note that this additional value reduces the number of possible matches under given error bounds, as the match measure  $q_\theta$  considers the distance of the point coordinates *and* their orientations. Figure 20 illustrates the recognition problem for the 2D case using edgels.

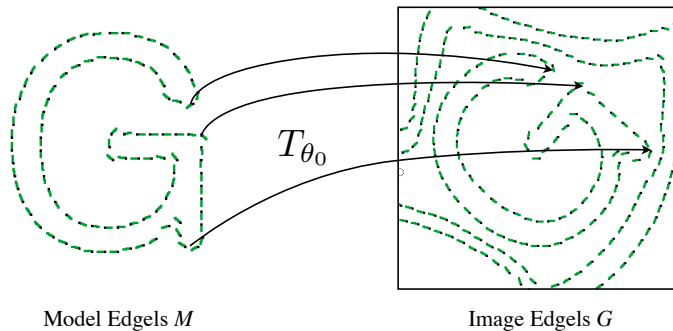


Figure 20: Illustration of 2D geometric matching of model edgels to image edgels. The orientation of the edgels are shown in green.

Many algorithms to solving this problem have exponential time complexity [13], or although fast, may miss good solutions (such as the generalized Hough algorithm [1]). The RAST algorithm [3]

on the other hand uses no heuristics, loses no solutions, and although it has exponential complexity in the worst case, it is usually much faster (for a more detailed look at RAST’s complexity, readers are referred to [3, 4]). RAST employs a branch-and-bound search over parameter space, that is the algorithm begins with a box in parameter space, that includes all transformations to consider. It then finds all correspondences between image and model features and evaluates the quality bounds of the matches resulting from the set of correspondences. If the upper bound on the best possible match is smaller than the minimum quality required for a match, or if it is smaller than another solution found already, the search is aborted. Otherwise, the current box is subdivided into smaller regions and the same process is recursively repeated. To evaluate which region to search next, all transformation candidates are maintained in a priority queue, sorted by the upper bound. The search terminates when either the quality bounds are within the required tolerance (match), or a pre-defined maximum number of iterations is exceeded (no match).

RAST has been successfully used for recognition of hand-printed digits before [5], making it a promising candidate for use in text recognition. In the previous section, we have seen that while template matching provides a fairly simple way of matching model glyphs to image locations, it is highly unsuitable when transformation space grows. RAST on the other hand, allows a search for matches over a large transformation space, including translation, scale and rotation of the model points.

To prepare the image and model for the search, we must transform each of them to a set of edgels. That is, to narrow down search space, each image point is associated with an orientation. More specifically, for the input image, we will simply use a subsampling of points along the image contours, obtained by edge detection. The associated orientations are evaluated from the gradient of the image, i.e. each point is associated with a normal vector pointing into or out of the enclosed component. Similarly, the contours of the model glyph, which are represented in a vector format, are subsampled into a set of contour points. The normals are computed and associated to the points. We are then ready to use RAST to find the optimal match between a set of model edgels and set of image edgels. Note that, while the normals of the model contours can be configured to always point to a certain direction (such as “into” the glyph), this possibility is not given in the image, where the normals point into or out of a character depending on its color (and the background color). Thus, RAST is configured to compare normal orientations disregarding the direction. The following parameters are used to configure the matching search:

- Transformation ranges  $(t_0, t_1), (s_0, s_1), (\alpha_0, \alpha_1)$ : Specify the search space of the matching process, spanned by the range of translation, scale and rotation. Obviously, these parameters have a large influence on the types of characters that can be matched, and on the overall performance of the algorithm.
- Match tolerance  $\epsilon_t$ : This value specifies the maximum distance allowed between an image edgel and an edgel in the model to be considered a match. Here, a distance is used that is proportional to the matching scale. That is, if a prototype is matched to the image at half scale,  $\epsilon_t$  will also decrease by one half.
- Angle tolerance  $\epsilon_\alpha$ : Specifies the maximum angle difference allowed between an image point and a point in the model to be considered a match. Angles may be compared with or without respect to orientation.
- Max Iterations: The maximum number of splits in search space before giving up on a solution.
- Tolerance: A threshold that determines the minimum size of a search space subdivision before it is accepted or rejected as a solution.

For each glyph model, RAST is run on the input image, using a certain set of transformations. This may either be the set of transformations confined to a hypothesis location, or the set of all possible transformations over the entire image. The transformation space is specified by the lower and upper bounds on translation, scale, and rotation. Each match is assigned a quality of match, where a combination of the match ratios  $p$  and  $r$  is chosen:

$$p = \frac{\# \text{ matched edgels in model}}{\# \text{ total edgels in model}}, \quad r = \frac{\# \text{ matched edgels in image}}{\# \text{ total edgels in image}}.$$

The ratios are combined to a single score and using  $k$ -nearest neighbor classification, the label of the most frequent prototype is chosen. There are two methods of combining the ratios to a single score. The first is evaluated by simply taking the minimum of the two. Such a score degrades gracefully as more of the instance of a model in an image becomes occluded, or as spurious points are added to the image. However, while such a score may work well when operating on a single character bounding box, it is not suitable for detecting characters over an entire image. There are two reasons for this. Firstly, when matching over an entire image  $r$  will generally be a very small number and not in the range of  $p$ , thus making a combination of the two difficult. Secondly, without a fixed character bounding box small characters will generally obtain a lower  $r$ -value than large characters. For this reason a second match measure is used when performing RAST matching over an entire image. Given a set of character matching ratios  $\{(p_1, r_1), \dots, (p_n, r_n)\}$  the each  $r_i$  is normalized to  $\bar{r}_i$ , by

$$\bar{r}_i = \frac{r_i}{\sum_{j=1}^n r_j}$$

The same normalization procedure can be applied to the values of  $p_i$ . Then, the match score of a character  $i$  is given by

$$s_i = \begin{cases} \frac{\bar{r}_i + \bar{p}_i}{2} & , \text{ if } p_i > t \\ 0 & , \text{ otherwise} \end{cases}$$

where  $t$  is some threshold for the minimum model match ratio  $p$ . The normalization makes the combination of the ratios possible, while the threshold eliminates those candidates that do not match well. However, this does not eliminate the problem of higher  $r$ -values for large characters. Thus, when operating on the entire image multiple matching passes are run. After each pass the points of matching characters are removed from the image. The number of passes is then the number of expected text strings of varying sizes. Tests conducted on typical scene data have shown that a value of 2 is usually sufficient.

### 3.6 Character Hypothesis Filtering

Generally, the methods for character hypothesis generation described above are configured to produce too many character candidates rather than too few. In other words, they output a large number of false positives. For this reason, it is important to filter those candidates that are not true characters. At the same time it is important to avoid filtering out any true positives as they cannot be recovered in a later step. Character filtering is performed after all character hypotheses have been generated, and before they are grouped. The filtering techniques are most often based on a set of character heuristics. In this work, the following filtering techniques are used:

- *Minimum area*: A character must exceed a minimum size to be considered for grouping. Imposing a minimum area restriction removes small spurious boxes usually found in noise. Often a value is chosen which reflects the minimum size of a character required to produce reasonable recognition rates.

- *Maximum area:* On the other hand, occasionally large areas of the image are incorrectly classified as a character. For instance, the outline of a sign may be incorrectly interpreted as a 'D' or an 'O'. Thus, a maximum area filter removes those hypotheses that exceed some fraction of the total image area.
- *Aspect Ratio:* Characters of the latin alphabet usually have bounding boxes that fall within a fixed aspect ratio range. Hence, hypotheses with bounding boxes outside of a tolerated aspect ratio range can be filtered out. This value must be chosen carefully when dealing with rotated characters, as the typical aspect ratio of characters does not hold in this case. For instance, an 'l' rotated 90° has an unusually large width relative to the character height.
- *Minimum Score:* This filtering technique can be employed if character recognition was performed. If a recognition score falls below a certain threshold, the character is considered noise and filtered out.

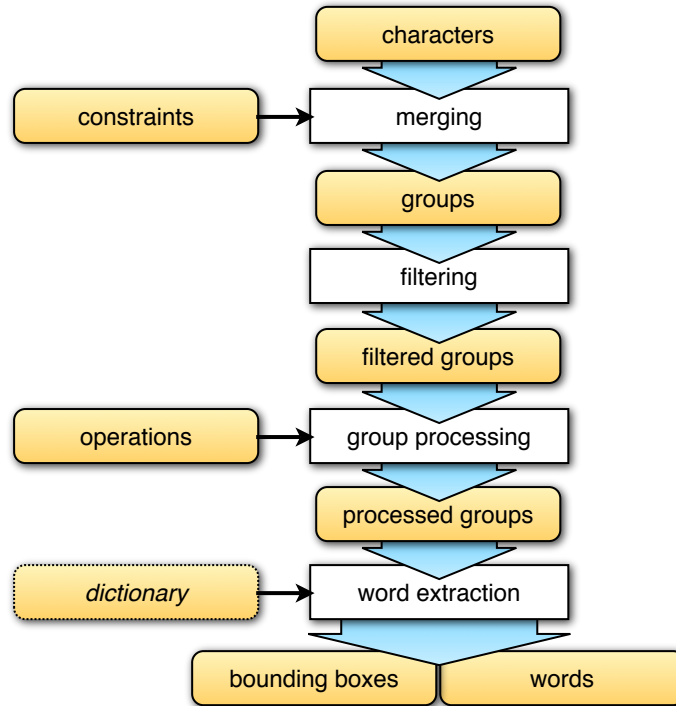


Figure 21: Overview of the grouping process, that converts character hypotheses to a set of words and bounding boxes.

## 4 Grouping

After the character hypotheses have been generated and filtered they are passed to the grouping module where they are converted to a set of words. The grouping module must determine which characters belong to the same group and reject character hypotheses that do not fit into any group. A number of steps are involved in the grouping process and Figure 21 gives an outline of these. In the following each step is discussed.

**Character Merging** The merge step converts a set of character hypotheses  $H$  to a set of groups  $G$ . At the beginning of the merge step, each character belongs to a distinct, single element group. To evaluate whether two characters are compatible for merging, a set of constraints  $C = \{c_1, \dots, c_n\}$  are employed, where each constraint  $c_i$  maps a pair of character hypotheses to a compatibility value:

$$c_i : H \times H \rightarrow \{0, 1\}, i \in [1, n]$$

All character hypotheses are considered pair-wise and merged to the same group, if they are compatible according to all constraints. For instance, in the extreme case, all characters are merged to the same group, while the other extreme is given when all characters remain in separate single-element groups. A number of constraints are used in this work, and a rough outline of each one is given below:

- *Maximum distance*: This constraint is satisfied, if the characters are no more than a certain distance apart. The distance is proportional to the character size.

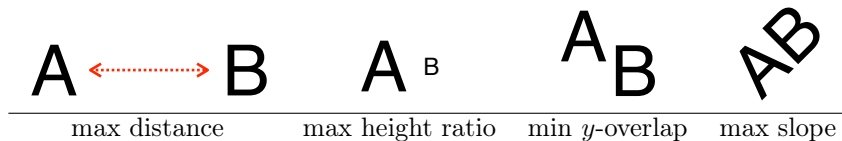


Figure 22: A set of example images showing grouping constraint violations, along with the name of the constraint each example violates.

- *Maximum height ratio*: Two characters are considered compatible under this constraint, if the height of the taller character divided by the height of the shorter character is less than a certain threshold.
- *Minimum y-overlap*: If the height of the bounding box intersection of two characters is no smaller than a certain threshold, the characters are considered compatible.
- *Maximum slope*: A pair of characters is considered compatible, if the absolute value of the slope of the line connecting the two characters is no larger than a certain threshold.

Figure 22 shows examples of character pairs that each violate one or more of the constraints. Note that by adding more constraints the set of recognizable character strings is reduced. For instance, if the slope constraint is used, text that is rotated or heavily distorted may not be grouped at all.

**Group Filtering** While the filters discussed in the previous section operated on character hypothesis, the filtering discussed here operates on character groups. In this work only a simple filtering mechanism is used, which filters groups containing less than a certain amount of characters. Usually, this value is set to 2, so that all single-character groups are removed.

**Group Processing** In the next step, the groups are processed and prepared for word extraction. In this work the following group operations are used:

- *Hole removal*: Using a number of heuristics, character hypotheses are removed from the group, that appear to be holes of larger characters.
- *Space detection*: Although the distance constraint may already separate text lines into words, the space width of word groups often varies, so that text with a small space width may be grouped together. These spaces can be detected by comparing the individual spaces with the mean space width.
- *Character sorting*: The characters within a group have no particular order. This operator sorts the characters in preparation for the word extraction step. While more complex sorting methods may be used, a simple sorting by  $x$ -position is sufficient for this work.

**Word Extraction** In the final step, words and bounding boxes are extracted from the groups. The bounding box is easily evaluated, and is simply the smallest box that encloses all of the character boxes. In order to extract a word from a group, the character hypotheses are processed in their sorted order, and for multiple conflicting hypotheses, a character decision is made, i.e. one of the hypotheses is accepted as the true character hypothesis. Multiple characters are in conflict with one another, if the overlap of the bounding boxes exceed some threshold. There are two methods of resolving character conflicts: The first simply selects the hypothesis with the

maximum recognition score. The second does a search over a (sub-)set of all possible character combinations, and chooses the word that best matches a dictionary entry. Here, the Levenshtein distance [22] is used to compare the word hypothesis with the dictionary entries. The value of the distance measure gives the number of insertions, deletions or substitutions required to convert one string to the other. Such a distance measure between strings is called an *edit-distance*. Each word hypothesis is compared to every word in the dictionary, and the word with the minimum edit-distance is then chosen as the output string. In case of a tie, the word that arose from the hypothesis with the highest recognition score is chosen.



## 5 Experiments and Results

### 5.1 Tests on Synthetic Data

In order to compare the performance of character detection and recognition techniques, a challenging set of 6,000 synthetic color character images were rendered. Although real-world images are usually preferable to synthetic images, the latter allows for controlled, specifically tuned test scenarios. Specifically for this case, a test set of images was required, where each image contains exactly one character glyph. This allows performance measurement of the character detection and recognition techniques only, without having to deal with multi-character issues such as linked letters, or incorrect grouping of characters. (Section 5.2 deals with the recognition of entire words in real-world data). Apart from the single character constraint however, it was important to obtain a test set that was as realistic as possible. More specifically, the test set should simulate the following real-world phenomena:

- A character may appear in any color or size, and may be rotated or distorted by perspective.
- Characters are present in a large number of fonts, ranging from simple standard fonts to heavily stylized ones.
- Text may lie on complex backgrounds, and may be difficult to segment from the background.
- Often text does, in fact, lie on a single colored surface, which may be affected by lighting.

To generate images with these properties, a rendering engine was implemented that was capable of rendering distorted characters onto complex backgrounds. The engine was configurable to allow fine-tuning of the distortion parameters. The rendering pipeline can be summarized as follows:

1. Generate a background image of a random size. (The size range is configurable by the tester). The background is randomly selected to be one of the following:
  - (a) A single random color  $c$  with illumination simulation. Specifically, the illumination was simulated by a color gradient to a lighter or darker version of  $c$ . Figure 3 (top left image) shows an example image with such a background.
  - (b) A random background image that does not contain text. Here, a hand-selected set of 100 images (downloaded from Flickr) were used, along with image patches from the ICDAR training set. The patches were obtained by analyzing ICDAR annotations, and selecting those regions of the image that did not contain text.
2. Select a random character (from a given list of characters) for rendering at a random location in the image.
3. Distort the selected character by scale, rotation or perspective, depending on the test configuration.
4. Choose a suitable color for the character. As we would like the character to be somewhat distinguishable from the background (which is normally the case in realistic images), the color histogram of the background area in which the character will be rendered is extracted. The color with the minimum frequency is selected. If there is more than one such minimum, a random one is chosen.
5. Render the character.

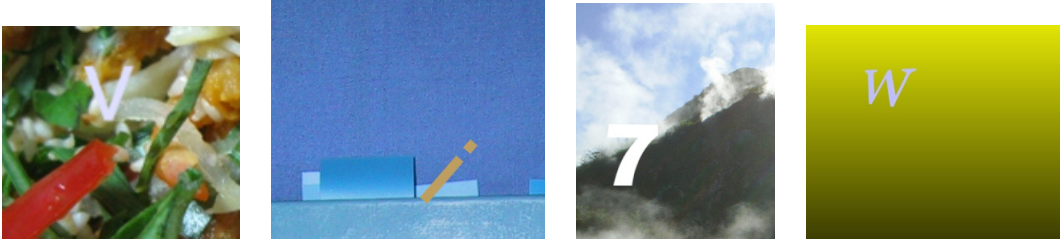


Figure 23: Examples of challenging synthetic test images.

A total of 6 sets of character images were rendered, each containing 1,000 images. The images in each of the 6 sets were generated using a different set of rendering configurations, in order to test robustness against various distortions. All images range in size from  $128 \times 128$  up to  $1024 \times 1024$ , not preserving this aspect ratio. An overview of the six configurations is given in Table 3, where  $s$  specifies the range of scale,  $r$  the range of rotation in 2D, and  $r_x, r_y, r_z$  the ranges of rotation in 3D. All unspecified ranges are implicitly set to zero. Note that the translation ranges are over the the image bounds for all test sets, and therefore omitted. Additionally, the table gives an overview of the abbreviations used for each set in the following discussion, and two example images from each set. Note that although the simulation of occluded characters or those that are linked to the background was not specifically implemented, these issues arise naturally from colorful backgrounds where the character color conflicts with at least some of the background colors. Figure 23 shows some of the difficult cases produced by our rendering engine. As the dataset should act as a “stress-test” of the discussed detection and recognition techniques, such a subset of challenging images is desirable.

### 5.1.1 Character Detection

The first test suite evaluates the performance of character detection using the bounding-box generation methods presented in this work (Section 3.3). Before the experiment setup and results can be discussed, a performance measure, that is adequate for the comparison, must be determined. The goal of this test is to investigate which detection methods are most likely to generate the bounding box of the true character present in each synthetic test image. For this, a *match measure*  $m_p$  between two bounding boxes  $b_1$  and  $b_2$  is defined by

$$m_p(b_1, b_2) = \frac{b_1 \cap b_2}{b_1 \cup b_2},$$

where  $b_1 \cap b_2$  is the intersection rectangle of  $b_1$  and  $b_2$ , and  $b_1 \cup b_2$  is the smallest box that encloses  $b_1$  and  $b_2$ . This value has the value one for identical rectangles, and zero for rectangles that have no intersection. Normally, this measure is used in text detection to estimate a real-valued *quality of match* for the returned boxes of the detection system. For the synthetic case however, which contain single characters only, the match measure is simplified to a binary measure called a *hit* (or its inverse function, *miss*), given by

$$h_\theta(b_1, b_2) = \begin{cases} 1, & \text{if } m_p(b_1, b_2) \geq \theta, \\ 0, & \text{otherwise} \end{cases}$$

for some minimum quality threshold  $\theta$ . In the following a value of  $\theta = \frac{3}{4}$  was used. For each test sample, a true bounding box  $b_t$  is given, and a set of evaluated boxes  $E$  are returned by the detection system. Then, the precision  $P$  and recall  $R$  is given by











Name	Fonts	Transformations	Example Images	
S	1	Size only: $s \in [24, \dots, 400]$		
SR	1	Size and 2D rotation: $s \in [24, \dots, 400]$ $r \in [-\frac{\pi}{2}, +\frac{\pi}{2}]$		
SP	1	Size and 3D rotation: $s \in [24, \dots, 400]$ $r_x, r_y, r_z \in [-\frac{\pi}{4}, +\frac{\pi}{4}]$		
FS	50	Font and size: $s \in [24, \dots, 400]$		
FSR	50	Font, size and 2D rotation: $s \in [24, \dots, 400]$ $r \in [-\frac{\pi}{2}, +\frac{\pi}{2}]$		
FSP	50	Font, size and 3D rotation: $s \in [24, \dots, 400]$ $r_x, r_y, r_z \in [-\frac{\pi}{4}, +\frac{\pi}{4}]$		

Table 3: Overview of the six synthetic image sets used for evaluation.

$$P = \frac{\sum_{b_e \in E} h_\theta(b_e, b_t)}{|E|},$$

$$R = \begin{cases} 1, & \text{if } \exists b_e \in E : h_\theta(b_e, b_t) = 1, \\ 0, & \text{otherwise} \end{cases}$$

These measures are then averaged over all test samples.

**Closed Contours** We begin by using closed contours to generate bounding box hypotheses. Here, a contour is considered closed if its endpoints are within a distance of no more than 1 pixel (in an 8-connected neighborhood). An analysis is performed to evaluate which edge detection method leads to the best bounding box generation results. Table 4 shows the precision and recall for the closed-contour generator on a subset of the synthetic data for the Canny edge detector and the Laplacian of Gaussian detector. Experiments were carried out using various settings for the Gaussian smoothing factor  $\sigma$ . Using the Canny detector, the recall drops with increasing  $\sigma$ , while the precision increases. This result is expected, as the more the image is smoothed, the fewer false positives we obtain. At the same time, a high smoothing factor may also break contours of true positives (as shown in the sample images in the same table). The LoG detector shows an overall poor precision. This is not surprising however, as the LoG detector *always* produces closed contours, making it an inadequate candidate for character detection by closed contours. The initial low recall of the LoG edges can be explained by the high amount of noise in the edge map, that often link character edges to background edges.

$\sigma =$	1.0	2.0	3.0	4.0	6.0
Canny	3% / 77%	18% / 77%	30% / 72%	36% / 69%	34% / 65%
LoG	0% / 30%	2% / 69%	3% / 71%	7% / 72%	7% / 70%

Table 4: Performance (average precision/recall) of closed-contour detection on Canny and LoG edge maps, using various values for the Gaussian smoothing factor  $\sigma$ . The edge output for a sample image (*left*) is displayed for each  $\sigma$ .

Although a higher Gaussian value may result in a better trade-off between precision and recall, we are more interested in high recall than precision. The justification for this, is that a missed contour is generally more difficult to generate, than to filter false positives. In the following experiments  $\sigma$  is set to 2.0.

In addition to the Gaussian parameter  $\sigma$ , the Canny edge detector uses two thresholds  $t_{hi}$  and  $t_{lo}$  for tracing with hysteresis, given by

$$t_{hi} = a_{hi} \cdot q_N,$$

$$t_{lo} = a_{lo} \cdot q_N,$$

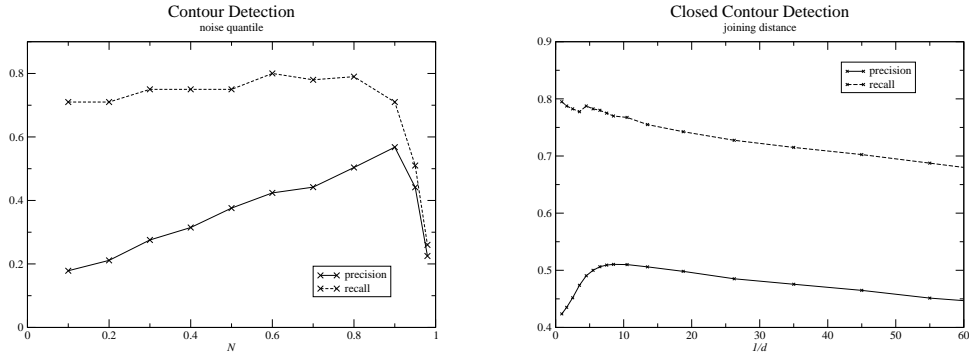


Figure 24: Average precision and recall of closed-contour detection on Canny edge maps with contour joining preprocessing step ( $\sigma = 2.0$ ). The left graph shows the results for various values for the expected noise  $N$ . The right graph shows the results for various settings of the joining distance threshold  $d$ .

where  $q_N$  is the quantile of edge magnitudes, that is considered noise. Here, we choose  $a_{lo} = 2.0$  and  $a_{hi} = 4.0$ . To evaluate a good choice for  $N$ , multiple experiments for an increasing value of  $N$  were conducted on a subset of the synthetic test data. The results are shown in Figure 24 (left). The graph suggests that a value of  $N = 0.8$  provides a reasonable trade-off between precision and recall. Although this may seem like a surprisingly high value, and indeed many of the contours are broken, due to the fact that we employ a joining step (set to  $d = 1/5$  in the graph), and are generally not interested in preserving the contour shape, but merely finding its position, this value suits the task well.

As the results just demonstrated, a higher smoothing factor and noise quantile is beneficial for precision, but may produce gaps in shape contours. Usually, these gaps are no more than a few pixels wide, whereas contours produced from background noise are often short, and have endpoints that are far from each other. Thus, it makes sense to apply a contour joining preprocessing step, where end-points that are sufficiently close to one another are connected. Furthermore, this joining threshold should be adaptive to the contour length, so that a long contour has a higher probability of merging with another contour than a short one. To find the optimal setting for the maximum joining factor  $d$ , a series of experiments were carried out with decreasing values for  $d$  (and  $N$  set to 0.8). The results are plotted in Figure 24 (right), and show that a value of around  $d = 1/5$  produces a high recall with a relatively small drop in precision.

Using Canny edge maps with  $\sigma = 2.0$ ,  $N = 0.8$  and a contour joining step with  $d = \frac{1}{5}$ , we still achieve a rather poor precision of around 50%, but on the other hand, a recall of around 72% (i.e. 72% of the bounding boxes are found by the detector). As no filtering step has been done in these tests, this high amount of false positives is not surprising. Though the recall shows that most character boxes are indeed found, we shall provide a quick analysis of where even a high setting for  $d$  does not lead to correct boxes. Figure 25 shows such a case, where the character is linked to the background, so that potential end-points of the character contour have become corner points connected to a background edge.

A summary of the performance of the closed contour detector on all test images is listed in Table 6. Note that the detector is fairly robust against any of the transformations applied to the characters.

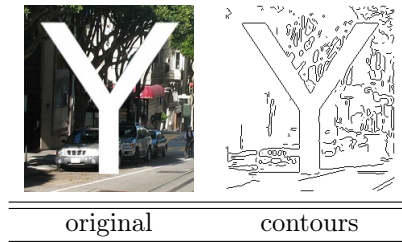


Figure 25: A case where closed contour detection does not lead to the correct character hypothesis, due to a links of the character contour to background contours.

**Character Strokes** Character bounding boxes can be detected by analyzing the properties of the skeletons obtained from the input image. Here, we discuss and evaluate two properties that are typical for character skeletons: Variation in stroke width and variation in color. We expect both of these to be low for character skeletons. Furthermore, two methods for extracting skeletons were implemented: The first obtains skeleton maps by morphological thinning, the other traces along ridge points obtained from the distance transform. In the thinning approach, the LoG operator was first applied to the image to obtain closed contours. Then, the set of connected components were extracted, and each one thinned (in parallel). In the ridge tracing approach, the distance transform was applied to the edge map (again obtained by the LoG operator), the ridges traced, and each connected skeleton subsequently pruned. As skeleton extraction tends to produce a large number of small skeletons in noisy areas, skeletons are ignored that have a length of no more than 8 pixels. These would otherwise be promoted to bounding box candidates, as very short skeletons tend to have a low variance in their properties.

Table 5 shows a comparison of the precision and recall obtained from skeleton analysis using the thinning and ridge-tracing algorithms. The tests were conducted using the stroke width variance (left in table), and the stroke color variance (right in table) as indicators for character locations. A skeleton was classified as a character skeleton, if the stroke width variance was below  $t_{SWV} = 0.5$ , or if the stroke color variance was below  $t_{SCV} = 64$  respectively (the color ranges from 0, ..., 255 for each channel) . As the results show, both methods perform about equally well in terms of recall. However, the ridge tracing technique produces significantly more false positives. This is somewhat suprising as the ridge tracing method produces fewer skeletons in noisy areas (recall that the skeletons are pruned after ridge tracing). However, the thinning method tends to produce so many skeletons in noisy areas that these are all interconnected, and are immediately discarded as potential character locations. Figure x shows a comparison of two images and the skeletonization output of both methods. However, it should be noted, that the ridge tracing method is much more

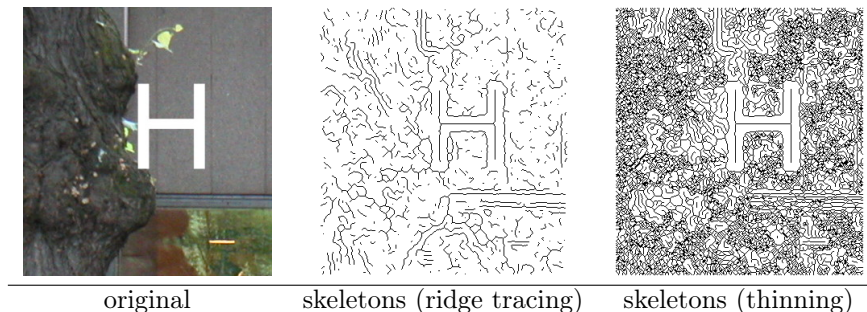


Figure 26: Comparison of skeletons obtained by ridge tracing (*middle*) and morphological thinning (*right*).

	Stroke Width Variance	Stroke Color Variance
Thinning	32% / 60%	27% / 48%
Ridge Tracing	3% / 59%	3% / 48%

Table 5: Precision and recall of the stroke width variance detector, using two skeletonization approaches: thinning and ridge tracing.

	S	SR	SP	FS	FSR	FSP
CC ( <i>no joining</i> )	31% / 64%	30% / 65%	30% / 64%	30% / 65%	28% / 63%	28% / 63%
CC ( $d = \frac{1}{5}$ )	48% / 75%	47% / 74%	48% / 73%	51% / 77%	47% / 74%	46% / 73%
SWV	33% / 69%	30% / 68%	24% / 58%	33% / 54%	26% / 52%	20% / 44%
SCV	23% / 66%	24% / 68%	25% / 66%	36% / 65%	25% / 61%	26% / 57%

Table 6: Summary of performance results (precision / recall) of character detection, without bounding box filtering.

efficient than the thinning method, which usually requires numerous passes over the image.

The threshold chosen for the stroke width (or color) variance provides a trade-off between precision and recall. Obviously, a higher threshold leads to a higher recall, but lower precision. Figure 27 shows the precision and recall for various values of  $t_{SWV}$  and  $t_{SCV}$ , evaluated on a subset of the test data. Values of 0.3 for  $t_{SWV}$  and 128 for  $t_{SCV}$  seem to provide a reasonable trade-off. Using these thresholds, both methods were tested on the full synthetic test data. The results are summarized in Table 6. While the detectors are robust against rotation, the error rate for the detector based on stroke width variance increases for characters distorted by perspective, and varied by font. The first can be explained by the fact that the stroke width varies more and more with growing perspective distortion, while the second can be explained by stylized fonts that use strokes of different widths. The detector based on stroke color is not affected by this problem (though it may be in real-world data, where lighting conditions could alter the color appearance).

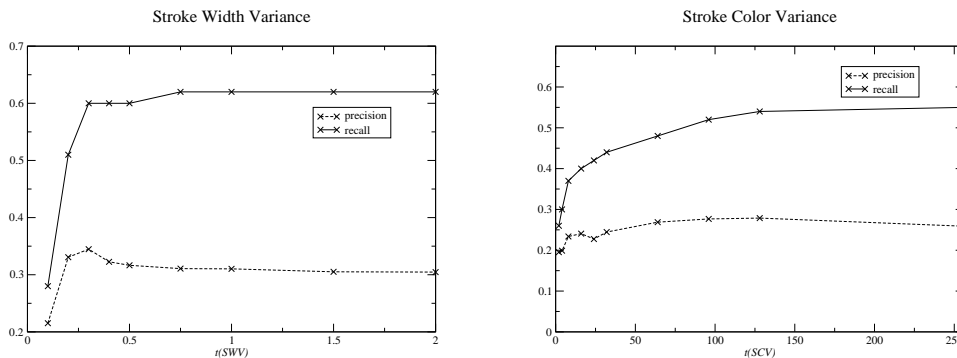


Figure 27: Performance plot of the stroke width/color variance detectors for varying threshold values.

**Filtering False Positives** As Table 6 shows, closed contours and character stroke detectors are capable of achieving reasonable recall rates. The precision, however, has been comparatively poor for both methods. One way to improve precision is to filter out any bounding boxes with properties

	S	SR	SP	FS	FSR	FSP
CC	56% / 75%	54% / 74%	53% / 73%	57% / 77%	54% / 74%	53% / 73%
SWV	43% / 69%	43% / 68%	37% / 58%	40% / 54%	34% / 52%	26% / 44%
SCV	36% / 66%	34% / 68%	40% / 66%	45% / 65%	34% / 61%	35% / 57%

Table 7: Filtered localization results (precision / recall)

that suggest they are not actual character bounding boxes. Table 7 shows the improved precision rates, when adding the following filters to the processing pipeline:

- A minimum area filter with a threshold of 64 pixels,
- a maximum area filter set to 1/8th the size of the image,
- an aspect ratio filter with the maximum *width/height* set to 1.6.

Overall these filters show an increase of precision by 6% – 15%, without affecting recall. Although more strict filtering values would lead to higher precision values, this would also result in a lower recall.

**Summary** While both approaches to text detection perform reasonably well, the closed contour approach has the overall higher scores, and given its simplicity is the more convincing method of the two. Inspecting the results showed that while the stroke detection methods may find bounding boxes that the closed contour method does not (and vice versa), combining both approaches showed only little gain in recall ( $\sim 3\%$ ), and a higher loss in precision. The recall rate of around 75% means that 1 in every 4 characters is missed in detection. That is, this error is inherently included in all recognition methods that operate on the detected bounding boxes.

### 5.1.2 Character Recognition on Known Locations

This section provides a performance comparison of the character recognition techniques discussed in the previous sections. Here, we are only interested in the character recognition performance, and not in the ability to detect characters. For this reason, a character detection step is not performed, but instead the ground-truth bounding boxes are given directly to the character recognition module. Performance is again tested on the synthetic test set of 6,000 images with various distortions. For performance evaluation, the evaluated character  $c_e$  is compared to the true character  $c_t$ . A hit is assigned if and only if  $c_e = c_t$ . Overall performance is then given by the number of misses divided by the number of test samples. This ratio is called the *error rate*.

All recognition methods require a training step on a set of labeled character images. These are then used to train a classifier, or used as prototypes in the  $k$ -nearest neighbor search. In order to test the influence of fonts used in training, multiple character image sets were generated. The first set consists of alphabetical and numerical characters (60 in total), rendered in the Arial font (120 point) only. Arial was chosen, as it is a relatively simple font, containing no serifs or other decorative elements. The second set consists of roughly 3,000 alpha-numeric character images, rendered in various fonts (120 point). This set is used to analyze performance increase when multiple fonts have been used for training. Finally, a third set was rendered consisting of about 12,000 images, containing rotated characters of multiple fonts and sizes. This set was not used for training any of the  $k$ -nearest neighbor methods as it would result in models too large to be









sample	contour	skeleton
<b>K</b>		
<b>M</b>		
<b>6</b>		

Table 8: Three examples of the training samples used for training the character classifiers. The middle and right columns show the extracted contours and skeletons.

efficiently used in the classifier. Instead, the set was used for training classifiers with a compact model, such as those of decision trees.

A contour based recognition method loads the character images of one of the training sets, and applies the LoG operator for edge detection to each one. The edges are then converted to a vector representation. Skeleton based methods perform an additional skeletonization step after the edges have been extracted from each character image. Again, the skeletons are converted to a vector representation. Table x shows a few example character images from the multi-font training set, along with the extracted contours and skeletons.

**Template Matching** We begin our analysis of character recognition techniques with template matching. In the most simple approach, the color variance of the image pixels that coincide with the template mask is used as a quality measure for a match. The problem with this measure is that templates that only match a part of the character may produce a high match quality. In a slightly more advanced approach, the match quality is determined by the distance of edges in the image to the edges in the template. Unlike the color variance approach, however, this technique depends on the edge detection method, and may fail if edge detection has led to incorrect or noisy edges. Table 9 shows the error rates of template matching using both approaches, using prototypes from the Arial font only. While the quality measure using edges outperforms the more simple color variance approach in the single-font test set (S), the color variance approach shows better results in the multi-font case (FS). This suggests that the color variance approach is in fact more robust to changes in the font. Furthermore, the results show that template matching is not suitable for rotated characters or those distorted by perspective. Generally, template matching also shows a large performance loss when matching to characters of an untrained font. Finally, it is interesting to ask why 7% of the most simple set (S) are classified incorrectly. Inspecting the incorrect cases shows that the remaining error can be mostly explained by glyph ambiguities, such as confusing a '0' for an 'O', or a '1' for an 'l'.

	S	SR	SP	FS	FSR	FSP
TM (color variance)	12%	86%	74%	45%	91%	80%
TM (edge energy)	7%	82%	61%	51%	88%	78%

Table 9: Comparison of template matching results (error rates), using the color variance, and edge energy measures (Arial font only).

To allow prototypes of multiple fonts, the best match measure is extended to a  $k$ -nearest neighbor measure. To evaluate a good choice for  $k$ , the template matching approach (using edge energy) was trained on the entire glyph training set, and tested on a subset of the synthetic data for multiple values of  $k$ . As Table 10 shows,  $k = 1$  returned the best overall results. This suggests that the

	S	FS
k=1	14%	23%
k=3	17%	34%
k=5	19%	38%
k=10	32%	47%

Table 10: Template matching performance (error rate), using  $k$ -nearest neighbors classification and the edge energy match measure on the S and FS test sets. A value of  $k = 1$  produces the overall best results, suggesting that classifying by best match outperforms a standard  $k$ -nn approach.

prototypes aggregated typically match a single font variant only, so that an evaluation by voting is not adequate. Using  $k = 1$ , template matching was evaluated on the entire synthetic test set, using the edge energy and color variance matching measures. The results are summarized in Table 21.

Given its sensitivity to any form of distortion, template matching is obviously not a good candidate for text recognition of characters in random imagery. However, it does provide us with a baseline performance of a naive solution to the text reading problem.

**Orientation Histograms** Given a contour, an orientation histogram is created from the orientations of the points that lie on this contour. The histograms are made invariant to rotation, by normalizing the values to the angle of the first principal axis. Here, contours were extracted from the bounding boxes using the LoG edge detector. Two distance measures were tested for histogram comparison: The standard Euclidean distance, and the Jensen-Shannon Divergence. Overall the Euclidean measure showed higher accuracy (40% error rate), compared to the JSD measure (61% error rate). Both methods however show surprisingly high error rates.

The poor results suggest that orientation features are not a reasonable basis for character classification. Compared to simple template matching, it is more robust against rotation, though overall it falls significantly short of the results obtained using template matching. While one conclusion may be that orientation distribution simply is not discriminative enough to classify the full range of characters, another origin of error may lie in the fact that only closed contours are used. We have stated before that the largest closed contour is selected for feature extraction. Of course, if an incorrect contour is chosen, or none at all, then we can expect classification to fail as well. In fact, if we restrict our analysis to only the correctly identified contours, we find that performance increases dramatically.

Table 11 gives a comparison between the overall error rate, and the error rate for the set of correct contours only. That is the error rate was calculated for those cases only, where the contour selected by the recognition algorithm was indeed the outer contour of the true character. (Note that although the ground-truth contours are not given, the cases where the correct contour is chosen was estimated by selecting those cases where the bounding box of the extracted contour is nearly identical to the ground-truth bounding box). Although the LoG edge detector exclusively produces closed contours, the correct contour identification error rates in Table 11 suggest, that performance is similar to Canny closed contour identification. The reason for this is that the LoG operator usually fails to produce a correct contour in the same cases the Canny edge detector fails. Furthermore, the results show that a percentage of the classification error can indeed be attributed to the contour identification.

A summary of the orientation histogram performance is given in Tables 20 and 21. Overall, the results show that orientation histograms alone are not a suitable feature for character classification.

	S	SR	SP	FS	FSR	FSP	Avg
Contour Error	29%	28%	28%	27%	27%	26%	27%
Overall Error	48%	65%	70%	56%	72%	73%	64%
Classification Error	30%	52%	60%	42%	61%	64%	52%

Table 11: Performance comparison obtained by orientation histogram classification. The first row gives the percentage of contours misclassified. The error rates for all test samples are given in the second row, while the third row restricts the analysis to those test samples where the correct contour was identified. A total of 1,471 prototypes were used for classification.

**Fourier Descriptors** While the orientation histograms from the previous section use contour information obtained from the spatial domain, Fourier descriptors are contour features obtained from the frequency domain. More specifically, the largest contour in a given bounding box is subsampled into  $m$  equi-distant points, which are transformed into the frequency domain using the discrete Fourier transform. The descriptors are normalized to obtain invariance against translation and scale. To obtain starting point and rotation invariance, two approaches are tested: The first uses the coefficient magnitude only, while the second applies normalization operations in the spatial and frequency domain as explained in Section 3.4.2. Finally, the first  $n$  complex descriptors are extracted (i.e. the descriptors  $a(\lfloor -n/2 \rfloor), \dots, a(0), \dots, a(\lfloor n/2 \rfloor)$ ), resulting in  $2n$  real feature values. The contour is then classified by a  $k$ -nearest neighbor search over the prototypes, using the Euclidean distance measure.

In our experiments, we subsample contours into  $m = 256$  points. To evaluate a good choice for the number of descriptors to consider in classification, we conducted a series of test runs for various values of  $n$ , on a subset of the test data (single font only). The results of these experiments are plotted in Figure 28 (left). They show that minimal error rates are already achieved for  $n \approx 15$ . Furthermore, the plot shows that although descriptor matching by normalization performs better for low values of  $n$ , their performance is nearly identical for any  $n > 12$ . However, the error rates for both methods reach a lower bound at around 34%. This bound suggests that the error is caused not by the Fourier descriptor classification itself, but by the contour extraction step before. That is, the lower bound at 34% can be explained largely by the incorrect selection of the closed contour. Figure 28(right) shows classification performance restricted to the set of correctly identified contours. The graph shows that the isolated Fourier descriptor classification performance can indeed reach error rates below 10%. Inspecting the remaining incorrect cases shows that the misclassifications can again be mostly explained by glyph ambiguities.

Table 12 shows the performance results (restricted to the set of correctly identified contours) of the Fourier Descriptor classifier trained on multiple fonts, and tested on the entire synthetic test set. The results show a performance loss involved when taking the descriptor magnitude only: Both the number of prototypes for training as well as the error rate show a significant increase, compared to the descriptor normalization approach.

	#proto	S	SR	SP	FS	FSR	FSP	Avg
FD (Mag)	1054	25%	23%	36%	31%	27%	41%	31%
FD (Norm)	674	9%	15%	19%	13%	16%	23%	16%

Table 12: Performance (error rate) of Fourier Descriptor classification trained on multiple fonts on the set of correctly identified contours only.

The overall performance of the Fourier descriptors can again be found in Tables 20 and 21. Although both the orientation histogram and Fourier descriptor classifiers operate on the contour only, the latter shows significantly lower error rates.

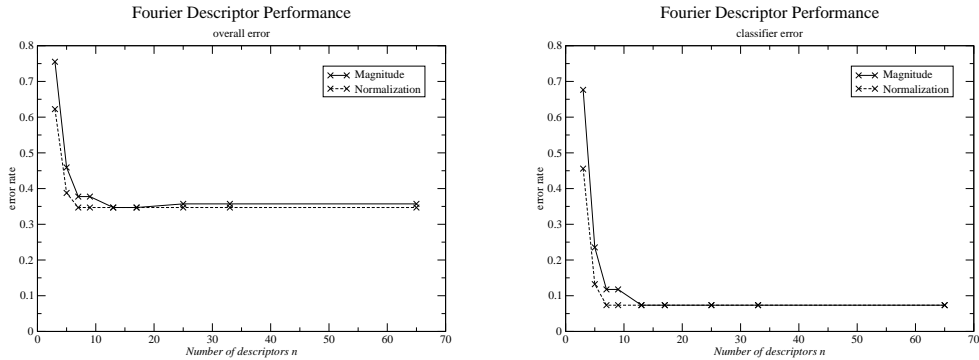


Figure 28: Error rate of classification by Fourier Descriptors using the first  $n$  descriptors only. The error rates for all test samples are shown on the left, while the graph on the right restricts the analysis to those test samples where the correct contour was identified.

**Skeleton Classification** The skeleton of a character captures the structural properties of its shape. A set of discrete features are extracted from the skeleton and used for classification. These features are invariant to translation, scale, and to a certain degree, rotation and perspective distortion. The skeleton properties were extracted from the large character training set, containing 12,000 character samples of various fonts, scale and rotation. Four classifiers were trained and tested on these samples. Table 13 shows the classification performance using 10-fold cross-validation for each of the classifiers.

classifier	correctly classified
Naive Bayes	54%
C4.5 (decision tree)	72%
CART (decision tree)	71%

Table 13: Classification performance (percentage of correctly classified instances) on the 12,000 character image set using 10-fold cross-validation.

The Naive Bayes classifier performance is far below that of the decision trees. These results are expected, due to the different nature of the feature values, where some reflect total counts (number of end-points, junction, etc.), and others normalized frequencies (such as the orientation frequencies). Decision trees are generally a good choice for dealing with features of different domains. The remaining error of 28% incorrectly classified instances shows that there is still a considerable amount of ambiguity among the features. A look at the confusion matrix showed the following frequent reasons for misclassification:

- As classification was performed using case-sensitive comparison, so that confusing a “c” with a feature-identical “C” was counted as a misclassification.
- Often characters with nearly identical features, such as “O” and “D”, or “8” and “B” are confused.
- The variation of scale and rotation in the training set increases the level of ambiguity, so that characters such as “C” and “U” may have identical features.

	S	SR	SP	FS	FSR	FSP	Avg
Classification Error	11%	11%	26%	22%	25%	35%	22%
Overall Error	33%	30%	42%	42%	45%	52%	41%

Table 14: Recognition performance (error rate) of the skeleton feature classifier (using a C4.5 decision tree) on the synthetic test images.

Though these misclassifications result in an initially higher error, these common confusions can usually be detected in a later stage and corrected. These correction mechanisms may use methods such as confusion heuristics or dictionaries.

Table 14 shows the performance results of the skeleton feature classifier (using a C4.5 decision tree) trained on the 12,000 character samples, and tested on the entire synthetic test set.

**Summary** While some of the methods discussed in this section are capable of recognizing contours or skeletons with error rates as low as 16% on the entire synthetic data set, they rely on the assumption that the contour or skeleton they operate on, are correct. The overall error rates show that this assumption often does not hold, even though the correct bounding box was given. Thus, to achieve reasonable recognition rates on random imagery, more sophisticated contour or skeleton extraction methods must be found. However, while there are certainly possibilities to improve character contour detection, it is unlikely that zero error can be obtained.

Template matching on the other hand, provides an alternative to the contour and skeletonization approaches: It does not require a previous segmentation step like the other methods. Instead, (using the edge energy method) every edge point in the image contributes to the match, regardless of whether it is part of the character or not. The downside of this method is its high sensitivity to rotation and perspective distortion. In the next section, RAST matching is used for character recognition on known bounding boxes. It combines the advantages of template matching, and the contour based methods, requiring no segmentation step while providing robustness against translation, scale and rotation.

**RAST Matching** In this section character recognition using the geometric matching technique RAST is inspected. As in template matching, no closed contours or other segmentation step is required, and instead RAST operates directly on the edge data. The edges in the image are subsampled to obtain a set of points. Orientations taken from the edge map are added to the points to obtain edgels. Then, every prototype is matched to the edgels within the character bounding box using RAST.

We will begin with a simple configuration of RAST, and iteratively add further parameters to transformation space. In each iteration, performance will be evaluated, and the misclassified characters inspected. This will allow us to learn why RAST is misclassifying certain samples, and open room for parameter tuning. Another reason for this more detailed inspection is that we expect RAST to perform very well: After all, RAST is capable of delivering the optimal match between the prototype points and the image points. It is therefore interesting to analyze in which cases despite this optimality RAST does not lead to the correct hypothesis.

We begin our experiments by setting all RAST transformation ranges to zero, i.e. RAST will perform a single match iteration for each prototype without applying any transformations. For now, we will also omit the multi-font cases, and first focus on the single font classes (Arial) only. Note, that there is an implicit translation and scale modification of the prototype points so that they fit inside the bounding box (preserving aspect-ratio of the original prototype bounds). The

results of applying this configuration of RAST to the synthetic test data are given in Table 15. As no transformation is applied, recognition performs poorly on the rotated characters. However, performance on the non-rotated characters are comparable to the template matching results.

RAST	S	SR	SP	FS	FSR	FSP	Avg
error rate	9%	60%	22%	22%	71%	38%	37%

Table 15: Error rates of character recognition using RAST matching with no transformation. As expected, rotated characters are not recognized well.

In order for RAST to find matches for rotated characters, it may seem sufficient to set a reasonable range for the rotation parameter  $r_\alpha$ . RAST will then find the best match for a prototype rotated within this range. As Table 16 shows, setting  $r_\alpha = [-\pi/2, +\pi/2]$  does in fact increase robustness against rotated characters. However, performance is still lower on rotated characters than on upright ones.

RAST+R	S	SR	SP	FS	FSR	FSP	Avg
error rate	9%	20%	16%	20%	46%	32%	24%

Table 16: Error rates of character recognition using RAST matching with rotation transformation.

The reason for this relatively poor performance is that the rotated characters introduce a number of challenges. Specifically, when dealing with rotation the following issues arise:

- *Shift of center*: The center of a bounding box of a rotated character may not coincide with the center of the character. Thus, if a matching algorithm fixes the prototype position at the bounding box center, the prototype points may show a translation error with respect to the image points. Figure 29 (left) shows a scenario where RAST was not capable of matching the character as the bounding box center was too far away from the character center.
- *Scale change*: The bounding box size of a rotated character is usually larger than the actual rotated bounding box of that character. Thus, a matching algorithm that fixes the scale of the prototype to the scale of the bounding box, may be attempting to match an oversized prototype. Figure 29 (right) shows such a case.
- *Noise*: As the bounding box of a rotated character is usually larger than the actual character, the amount of background points within the bounding box may be substantial and even exceed the amount of character points. A matching algorithm may then end up matching the prototype to noise, instead of the actual character contour.
- *Dimensionality*: Of course, when allowing any kind of transformation, more (incorrect) possibilities for matching are introduced. For instance, when allowing no transformation at all, there are only  $p$  possible matches, where  $p$  is the number of prototypes. If we now allow matching to  $n$  distinct rotation angles, the match possibilities expand to  $np$ .

While the last problem is inherent with geometric matching, it should be noted that there is a trade-off between the first two problems and the last one. That is, the translation and scale error can be tackled by using appropriate translation and scale ranges in matching. However, these additional degrees of flexibility will also result in a higher amount of false matches. Of course, false matches are all the more likely if a lot of noise is present in the matching region. Thus, in order to

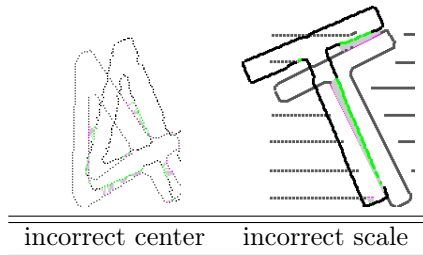


Figure 29: Examples of two difficulties introduced by rotation. The points in gray are the image points, while the black points are the points of the matched template. The matching points are highlighted in green (for the model), and magenta (for the image). The left image shows incorrectly assumed center coordinates, while the right shows an incorrect scale.

keep the likelihood of a false match at a minimum, it makes sense to first look at ways of reducing noise.

Recall that both the Canny and LoG edge detectors apply a Gaussian smoothing operator to the input image. Obviously, setting the smoothing factor  $\sigma$  to a high value results in less noise within in the edge map. On the other hand, a high amount of smoothing removes details from contours, which can be especially problematic for small characters. If we assume that the bounding boxes of the character are known, we can employ an *adaptive edge detector*, that applies a smoothing factor proportional to the image size. That is, instead of applying edge detection to the whole image beforehand, we extract the edges separately for each location.

A second parameter that can be tuned to avoid noise, is the edge strength threshold used for tracing (recall that two are used for the Canny edge detector). Setting this parameter to a high value removes any contours that originated from weak edges in the image. While in many cases weak edges are in fact an indicator for noise, sometimes the contrast between a character and its background is low, and hence, the edges weak. It makes sense to model the tracing threshold  $t$  proportional to the contrast of the pixels within the bounding box. It may, however, be difficult to find a suitable definition of contrast. One method, that provided the overall best results, is to simply begin with a high threshold  $t_{hi}$ , and check if a reasonably large contour was produced. If not, a lower threshold  $t_{lo}$  is used.

In Table 17 below, the results for RAST with rotation transformation are given - this time using the adaptive edge detection methods. As the error rates show, performance has increased significantly compared to the non-adaptive method before. Most of the remaining incorrectly classified samples, are either due to character ambiguities, or arise from the translational and scale errors of the bounding boxes.

RAST+R	S	SR	SP	FS	FSR	FSP	Avg
error rate	7%	15%	11%	15%	43%	28%	20%

Table 17: Error rates of character recognition using RAST matching with rotation transformation and adaptive edge detection.

To overcome these problems, we begin by specifying a scaling range of  $r_s = [0.75, 1.0]$ , where a value of 1.0 specifies the scale of the model fitted to the bounding box. As the bounding box is always at least the size of the character, the range of scale has an upper bound of 1.0. Finally, we specify the translation ranges  $r_x = [x - \frac{1}{4}w, x + \frac{1}{4}w]$ , and  $r_y = [y - \frac{1}{4}h, y + \frac{1}{4}h]$ , where  $w$  and  $h$  are the width and height of the character bounding box. The results of RAST using these transformations are given in Table 18. Using rotation and scale transformation, the RAST detector shows the

	S	SR	SP	FS	FSR	FSP	Avg
<i>Fourier D.</i>	34%	36%	42%	55%	54%	59%	47%
RAST+RS	5%	11%	10%	15%	31%	26%	16%
RAST+RST	4%	9%	17%	34%	37%	43%	24%

Table 18: Error rates of character recognition using RAST matching (with adaptive edge detection), and rotation (R), scale (S) and translation (T) transformation. The Fourier descriptor error rates are shown for comparison.

RAST	S	SR	SP	FS	FSR	FSP	Avg
<i>Fourier D.</i>	32%	36%	40%	33%	40%	43%	37%
RAST+RS	6%	15%	11%	8%	20%	11%	11%
RAST+RST	5%	12%	19%	20%	27%	28%	19%

Table 19: Error rates of character recognition using RAST matching, trained on multiple fonts with rotation (R), scale (S) and translation (T) transformation. The Fourier descriptor error rates, trained on multiple fonts, are shown for comparison.

overall best results. Enabling translation on the other hand, shows an increase in the error rate. We assume the higher error is due to the high dimensionality, i.e. by enabling all transformation options, false hypotheses may match well when transformed in rotation, scale, and translation. The higher error rates for the more stylized font sets support this assumption. In comparison to the Fourier Descriptors, RAST shows greatly improved performance. It is interesting to note that the classification performance of the Fourier descriptors on correctly identified contours only was similar to the RAST performance shown here. This fact gives an impression of the scale of improvement made by omitting a separate segmentation step.

At this point, it should be noted that RAST performance on unknown fonts is already quite remarkable. While training on multiple fonts is likely to increase performance on the font sets even more, the additional prototypes may also cause more confusion. Table 19 shows the recognition results for RAST matching, trained on the font training set, using rotation, scale and translation transformations. While performance increases on the multi-font test sets, the single font sets show a slightly higher error rate than when using Arial prototypes only. This is expected, as the additional prototypes increase the room for false matches.

Finally, Figure 30 shows two cases which highlight the strengths of using RAST. Note that in all these cases, contour based methods would have failed.

**Summary** The low error rates of the RAST matching method show that omitting a segmentation step can lead to a drastic improvement in recognition rates. In turn, the comparatively poor performance of the previous methods show just how severely detection errors influence the recognition result. Tables 20 and 21 summarize the results obtained in this section.

While the ground-truth bounding boxes were used in place of a detection step in this section, the next section deals with character detection and recognition, given no prior character location knowledge. Just as in this section, the performance of segmenting the image into character regions before recognition is compared to recognition on the entire image using prototype matching and no previous segmentation step.



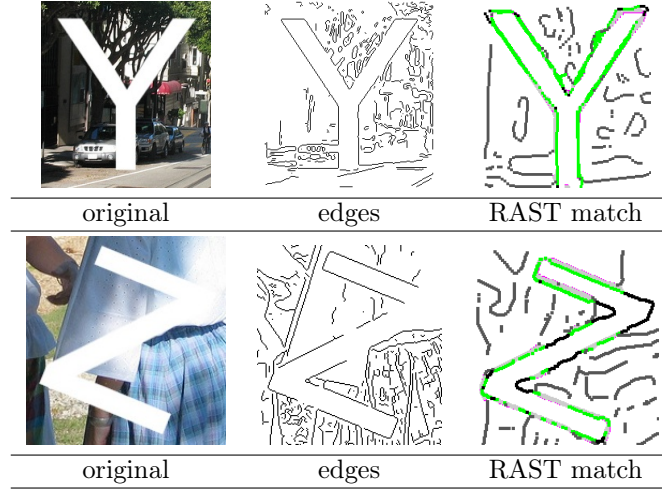


Figure 30: Examples of images, where RAST matching succeeds in determining the character class while feature classification methods fail.

	S	SR	SP	FS	FSR	FSP	Avg
Template Matching (EE)	7%	82%	61%	51%	88%	78%	61%
Orientation Histograms	42%	62%	69%	62%	76%	78%	65%
F-Descriptors (Mag)	37%	36%	42%	60%	61%	62%	50%
F-Descriptors (Trans)	34%	36%	42%	55%	54%	59%	47%
RAST	9%	60%	22%	22%	71%	38%	37%
RAST(RS)	5%	11%	10%	15%	31%	26%	16%

Table 20: Character classification results (error rates), using Arial font glyphs as training set only.

	#proto	S	SR	SP	FS	FSR	FSP	Avg
Template Matching (EE)	677	10%	79%	58%	20%	81%	65%	52%
Orientation Histograms	1471	48%	65%	70%	56%	72%	73%	64%
F-Descriptors (Mag)	1054	45%	43%	53%	47%	47%	56%	49%
F-Descriptors (Trans)	674	32%	36%	40%	33%	40%	43%	37%
Skeleton Classification	(tree)	33%	30%	42%	42%	45%	52%	41%
RAST (RS)	997	6%	15%	11%	8%	20%	11%	12%

Table 21: Character classification results (error rates), using multi-font prototypes.

	S	SR	SP	FS	FSR	FSP
CC+FD	47% / 63%	47% / 63%	42% / 56%	48% / 65%	45% / 62%	38% / 52%
CC+RAST	52% / 70%	50% / 69%	49% / 65%	56% / 70%	45% / 60%	41% / 63%
CC+Skel	43% / 53%	43% / 61%	38% / 50%	38% / 46%	31% / 42%	32% / 40%

Table 22: Character detection and classification performance (precision / recall)

	S	SP	FS	FSP
RAST	76% / 86%	46% / 74%	77% / 84%	57% / 61%

Table 23: Joint character detection and recognition (precision/recall) using RAST.

### 5.1.3 Character Detection and Recognition

Up to this point, the tasks of character detection and character recognition have been isolated. In the following, these two essential steps are combined, and tested on the synthetic test data. We begin by using the separate approach of text reading, where the character detection is a distinct step, followed by the character recognition step on the detected bounding boxes. Table 22 shows the results of applying closed contour recognition followed by a selection of the discussed character recognition techniques. The closed contour boxes were filtered using the size and aspect ratio filters. The results indicate that while RAST matching shows the best performance, all methods suffer from errors introduced in the character detection stage. Furthermore, it may be surprising that RAST does not outperform the other methods to the degree it did in the previous tests. However, as the detected text locations are the locations of the closed contours, RAST is just as likely to fail at recognition as the other contour based methods. The skeleton classification methods shows the overall lowest scores. Despite previous results showing a recognition performance of over 90% for skeleton classification, this only holds if the correct contour (for detection) and skeleton (for recognition) was chosen for feature extraction. Selecting any one of these incorrectly leads to an incorrect result, and explains the very poor performance of this method.

To avoid these detection errors, the character detection phase is eliminated altogether in the following, and RAST is directly applied to the entire image in the joint approach. As this may take a considerable amount of time to compute (see Table 24), RAST is configured to match using translation and scale transformations only. Hence, in the following, the rotated test cases are omitted. Furthermore, each of the sets is subsampled to 15% of the original image count. The RAST match score uses normalized recall values over all image points, and the number of matching passes was set to 1. Table 23 shows the results obtained when applying RAST to the synthetic test images. Note, that for the test sets with perspective distortion (SP and FSP), the tolerance value was increased to make up for the displaced character edgels. This results in a lower precision for these two sets. Overall the results outperform those from the separate approach. This suggests that geometric matching on the entire image without a detection step may indeed lead to a lower error rate. However, RAST matching tends to fail or show very poor precision in images that are very noisy. In such cases, the matching algorithm produces numerous false positives. Since a normalized recall measure over all image edgels is used, matching to incorrect image locations may distort the recall rates of all matches.

### 5.1.4 Summary

The methods described in this section can be split into those using a separate segmentation step, and those that apply recognition to the entire image. The results of the separate approach showed

method	time (in s)
CC+Fourier descriptors	0.3
CC+Skel (Thinning)	3
CC+Skel (Ridge tracing)	0.5
CC+RAST (RS)	24
RAST (no segmentation)	500

Table 24: Average amount of time, in seconds, the presented methods require to process one synthetic image.

that the errors introduced in the detection have a large impact on the performance of the recognition step. If the correct contours are identified, the Fourier descriptors provide the overall best results, while skeleton classification is slightly more robust against rotation. Character detection by closed contours outperforms the more complex stroke detection method, while a combination of the two leads to slightly higher recall with a drop in precision. Omitting segmentation steps entirely shows the overall best recognition performance. While template matching over an entire region is generally too expensive to compute if robustness against scale and translation is required, RAST matching provides a much more efficient alternative, and is configurable to match a number of transformations. However, as table 24 shows, despite its high efficiency, RAST matching still takes considerably more time than separate detection methods, and may not be feasible for most applications.

## 5.2 Word Recognition on Real-World Data

In this section, the task of character detection and recognition is extended to word detection and recognition. The discussed methods are tested on real-world photography containing text in natural scenery. In order to obtain results comparable to text detection work conducted by other authors, the tests are run on the ICDAR 2003 image sets. This publicly available dataset<sup>4</sup> was used in the recent text detection competitions ICDAR 2003 and ICDAR 2005. The dataset contains 258 images in the training set and 251 images in the test set. The images are full-color and range in size from around  $307 \times 93$  to  $1280 \times 960$ . Nearly all images contain text found in natural scenery, though some do not contain text at all. Furthermore, the ICDAR sets are known to be very challenging. Figure 31 shows a few examples of the challenges presented by the image set.

In addition to character detection and recognition, a grouping step is performed, to convert the set of character candidates to a set of word candidates. The output of the recognition system is then a set of words along with their bounding boxes. These bounding boxes are compared to the true bounding boxes, which are provided in the ICDAR dataset. Again, the match measure  $m_p$  is used, which for two boxes is given by the area of intersection divided by the area of the minimum bounding box containing both boxes. For each evaluated rectangle, the closest match is found in the set of true rectangles, and vice versa. Hence, the best match  $m(r, R)$  for a rectangle  $r$  in a set of rectangles  $R$  is given by

$$m(r, R) = \max\{m_p(r, r_0) \mid r_0 \in R\}$$

<sup>4</sup><http://algoval.essex.ac.uk/icdar/Datasets.html>



Figure 31: Sample images from the ICDAR train and test sets, displaying some of the difficult challenges they present.

Then, the definitions for precision and recall are

$$P = \frac{\sum_{r \in E} m(r, T)}{|E|},$$

$$R = \frac{\sum_{r \in T} m(r, E)}{|T|},$$

where  $E$  and  $T$  are the sets of estimated and ground-truth rectangles respectively. The standard  $f$ -measure is used to combine the precision and recall figures into a single measure of quality. The relative weights of these are controlled by a parameter  $\alpha$ , which is set to 0.5 to give equal weight to precision and recall:

$$f = \frac{1}{\frac{\alpha}{P} + \frac{1-\alpha}{R}}$$

These performance measures are consistent with the ones used in the official ICDAR evaluations 2003 and 2005 [27, 26].

The first experimental setup deals with text detection only, and should give a performance comparison of the two detection methods discussed in the previous sections. Table 25 shows the results obtained for the closed contour and stroke width variance detectors. Note that an improvement of the precision can be expected once character recognition is used with the possibility of rejecting incorrect candidates. Due to the higher performance of the closed contours both in  $f$ -measure and computational efficiency, the following experiments will all use this method in the text detection step.

	precision	recall	f-measure
Closed contours	20%	46%	26%
Stroke width variance	21%	2%	4%

Table 25: Comparison of the text detection methods on a subset (250 images) of the ICDAR data.

In the second experimental setup, separate detection and recognition is used, and the closed contour and stroke width variance methods are used for detection. Recognition is tested using the two

	precision	recall	f-measure	word recall
CC+FD	50%	42%	46%	33%
CC+RAST	49%	43%	46%	34%
Hinnerk Becker	62%	67%	62%	-
Alex Chen	60%	60%	58%	-
Ashida	55%	46%	50%	-
HWDavid	44%	46%	45%	-
Wolf	30%	44%	35%	-
Todoran	19%	18%	18%	-

Table 26: Separate character detection and recognition on ICDAR test data.

most promising techniques, namely Fourier descriptors and RAST matching with rotation and scale variation. The grouping method is configured with the following settings:

- Constraints: Characters are grouped that fulfill the distance constraint (1.0 times the character height), the  $y$ -overlap constraint (0.5 times the character height), and the  $y$ -size constraint (1.5 times the character height).
- Operations: Holes are removed, spaces detected, and the group characters sorted by  $x$ -position.
- Word Extraction: Words are extracted from the groups, using a standard English dictionary<sup>5</sup>, enriched with the words found in the ICDAR datasets.

Table 26 shows the results obtained for text detection and recognition on the ICDAR data sets, using separate detection and recognition. The detection step using closed contours gives an  $f$ -measure score of 0.46. The word recall gives the percentage of words matched in the entire set of true words. Here, recognition rates of 33% and 34% are obtained. As RAST matching is performed on the locations of the closed contours, its performance is only minimally better than that of the Fourier descriptors. The table shows some of the results obtained by the ICDAR 2003 and 2005 competition entries for comparison. These results suggest that closed contour detection is capable of performing reasonably well on the text detection task, and outperforms a number of the methods submitted to the competition. As none of the submitted ICDAR methods performed the task of text reading, a comparison cannot be given here.

Figures 32 and 33 show examples of where the separate approach used here produces good results, and which cases it fails. An analysis of the results showed, that the most frequent reasons for failure are:

- The closed contours did not lead to the correct character bounding boxes.
- The text was heavily stylized so that character recognition did not succeed.
- A word consisted of a single character only, and was thus eliminated in the group filtering stage.

Hence, in a final series of experiments joint detection and recognition using RAST is tested on the ICDAR set. As each such detection operation takes a considerable amount of time (see Table 24), three subsets of the ICDAR data are used, instead of the entire image set. The images selected for each set should specifically highlight the advantages or drawbacks when using joint detection and recognition. Specifically, each of the three sets contains images with the following properties:

<sup>5</sup>found on most UNIX-like systems under `/usr/share/dict/words`.



Figure 32: Example images where the separate approach (using closed contours and Fourier descriptors) produces good results. The ground-truth is shown in green, and the evaluated results in red. The returned words are given below the image.



Figure 33: Examples where the separate approach fails. Again, the ground-truth is shown in green, and the evaluated results in red.

1. Basic set (20 images): This set contains images for which the separate approach produced good results. This set should show whether or not a joint approach produces similarly good results.
2. Difficult set (35 images): This set contained images, where a separate detection and recognition method did not provide good results, even though the generated edge maps were reasonable, i.e. characters can be recognized in the edge map by a human observer. The failure of a separate approach was mostly due to linked characters or broken edges.
3. Noisy set (20 images): This set contains very noisy images, where characters are difficult to distinguish from noise - even for a human observer. Separate detection and recognition mostly fail at this task.

As the joint detection and recognition tends to produce more false positives, the grouping constraints are extended by a slope-constraint that omits any group of characters with a text-line slope more than 0.2.

**Basic Set** The results for the joint approach on the basic set are given in Table 27. While RAST shows a higher  $f$ -measure, it does not perform as well in word recognition as the separate approach. An analysis of the results showed that these errors are mostly due to errors in grouping, which must deal with a much larger set of character hypotheses than when operating only on closed contours. For instance, RAST often matches simple characters, such as “I” onto and in between the actual characters. The latter phenomenon was not detected when testing on the synthetic data, where only one character was present at a time. Often the grouping step generates incorrect character hypotheses interleaved with the correct ones. Furthermore, spacing may be incorrectly identified, so that words are split at the wrong locations. Hence, if a matching method like RAST is used for matching characters, a more sophisticated grouping and filtering is required to deal with the higher number of interleaved false positives.

	precision	recall	f-measure	word-recall
CC+FD	72%	78%	75%	81%
RAST	78%	83%	80%	66%

Table 27: Comparison of the separate and joint character detection and recognition on the basic subset.

**Difficult Set** The difficult set contains images, where the separate approach fails at locating atleast one of the text boxes. Table 28 shows a comparison of the separate and joint approaches on the difficult set. As the results show, using a joint approach can in fact improve detection and recognition scores on these images.

	precision	recall	f-measure	word-recall
CC+FD	31%	39%	33%	11%
RAST	53%	58%	52%	21%

Table 28: Comparison of the separate and joint reading approaches on a subset of the ICDAR test data.

Figure 34 shows two example images taken from the difficult set, that give insights to why the detection step fails for closed contours, and why RAST matching succeeds. Both edge-maps show



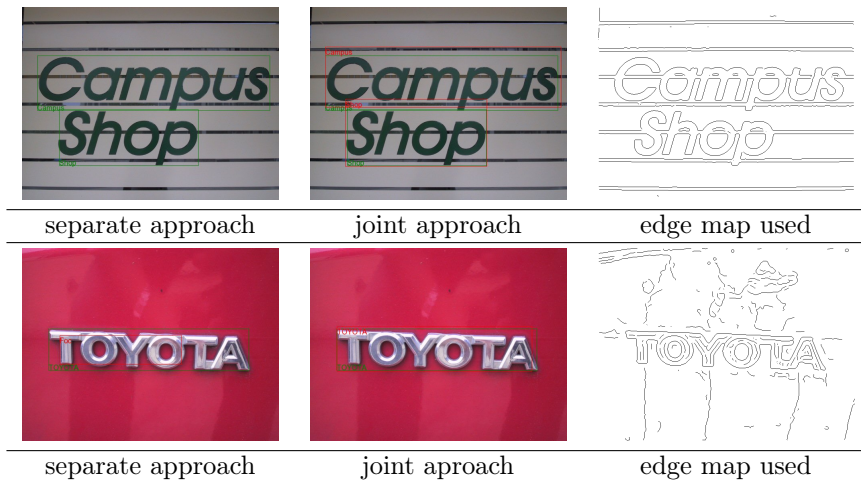


Figure 34: Example images from the difficult set, where the joint approach succeeds while the separate approach does not.

character contours that are not closed, but connected to the background or specular highlights. Obviously, closed contour methods do not perform well under such circumstances. The RAST algorithm on the other hand still finds enough edgels that contribute to the correct character prototypes to successfully recognize the words in both images.

**Noisy Set** Finally, the joint and separate approaches are tested on a very noisy set. These are usually images with strong specular highlights, that in turn lead to very noisy edge maps. Table 29 shows the results of applying the separate and joint approaches to the noisy image subset. While the separate approach produces acceptable scores, the joint approach fails to show reasonable performance for the image set. The reason for this is that the matching approach yields a high number of false positives, that are too numerous and spread over the image for the grouping step to handle correctly. Figure 35 shows a representative image from this set. Given that the text is difficult to read even for a human observer, the poor performance on such images is not surprising, and may not be an issue in applications, where cleaner images are expected.

	precision	recall	f-measure	word-recall
CC+FD	42%	36%	39%	17%
RAST	20%	9%	12%	0%

Table 29: Comparison of the separate and joint reading approaches on the noisy subset of the ICDAR test data.

### 5.2.1 Summary

As the results in this section show, even simple methods like closed contours lead to reasonable text detection performance on the very challenging ICDAR datasets. Overall, when using separate text detection and recognition steps, the Fourier descriptors provided the best trade-off between recognition rates and computational efficiency. While the synthetic tests showed that RAST matching on known bounding boxes can greatly improve performance, these improvements could not be measured here, as RAST was applied to the bounding boxes of the closed contours only.





Figure 35: Example of an image from the noisy set, along with the extracted edges.

Using a joint approach can improve detection and recognition results even further, though the higher recognition scores come at a cost of very long computation times. Furthermore, the joint approach fails when a large amount of noise is present in the image. In fact, since all of the approaches discussed here are based on edges, they all suffer a loss in performance when edge detection produces noisy results. In case the correct bounding boxes were indeed found, the character recognition works reasonably well. However, as a number of words in the image are missed in detection, this leads to an overall lower recognition score.

## 6 Conclusion and Perspectives

### 6.1 Conclusion

Automatic reading of text in natural scenery is a challenging problem. In this work, various methods were presented to accomplish the tasks involved. A text reading framework was implemented and discussed, which contained modules for preprocessing, character hypothesis generation, and grouping. A quantitative evaluation of the text detection and recognition methods these was given.

To generate hypotheses two main approaches were used: The separate approach, in which a character detection step precludes the recognition step, and a joint approach, where detection and recognition are performed in a single phase. Using closed contour detection as a means of text detection showed very satisfying performance on real-world data, and in case of the ICDAR datasets, even outperformed a number of entries of the previous competitions in terms of precision and recall.

In order to extract text strings from the discovered bounding boxes, a number of recognition methods were presented and evaluated. While some recognition techniques base their decisions on features extracted from the character contours or skeletons, the prototype matching techniques match a set of template characters to the image. In the case of feature classification, the Fourier descriptor method showed the overall best results. A downside to using such a technique is that in order for a classifier to produce the correct output, the outer contour of the character must have been correctly identified in the first place. Since a perfect contour extraction cannot be expected, it is most likely that a combination of features is best for classification.

Prototype matching on the other hand, does not require such a segmentation step, and can be directly applied to the image or image edges. In this work, a novel scene text recognition approach using RAST was introduced. Applying RAST to natural imagery showed potential in detecting even difficult to segment text. In the presence of a high amount of noise, however, RAST produced a high number of false positive character hypothesis that confused the subsequent grouping algorithm. Nevertheless, prototype matching remains an interesting topic for future work on joint text detection and recognition.

### 6.2 Outlook

Though numerous text detection and recognition methods have been implemented and evaluated in the context of this work, there are many more promising techniques to consider. The methods in this work are all based on character shape, represented by either the character contour or skeleton. A number of promising detection techniques based on texture have been proposed, and are worth comparing to the shape-based methods described here. Readers interested in such methods are referred to the Related Work section. Furthermore, the recognition techniques used here could be extended to include those that make use of textural properties.

However, while implementing additional detection and recognition methods may give insight into the compared performance of the text reading techniques, it is doubtful that any *one* of them will greatly improve performance. As suggested in [32], a combination of techniques is required to achieve overall satisfying results. In fact, currently we are investigating a probabilistic approach, in which multiple detection and recognition results are combined to a probabilistic model of character hypotheses over the entire image. The probability of a hypothesis is dependent on both the results obtained by detection and recognition, as well as the probabilities of neighboring hypotheses. We expect the performance of such a system to exceed that of the single-method approach used here.

As RAST shows potential for use in character recognition, it makes sense to conduct more research in this area. Specifically, additional transformations such as skewing or thickness variation

could be implemented and evaluated. Most importantly, methods are required that improve the robustness against noise. While one solution would involve tuning RAST to avoid matches in noisy areas, a different approach would involve extending the grouping mechanism to better deal with the large number of character hypotheses generated.

Finally, further evaluation of the current methods could give insights to the reading system performance on imagery of other domains. For instance, the current system could be evaluated on images of book covers or movie posters, where certain additional constraints may improve recognition. On the other hand, the ICDAR data used here did not include many examples of rotated or heavily perspective distorted text. In an additional dataset with more images of these types, robustness against these transformations could be better evaluated and improved.

## References

- [1] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. In *Pattern Recognition 13(2)*, page 111.
- [2] G. Borgefors. *Distance transformations in digital images*, page 344.
- [3] T. Breuel. Fast recognition using adaptive subdivisions of transformation space. In *Proceedings. 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.92CH3168-2)*, pages 445–51, 1992.
- [4] T. Breuel. Recognition by adaptive subdivision of transformation space: practical experiences and comparison with the hough transform. In *IEE Colloquium on 'Hough Transforms' (Digest No.106)*, pages 7/1–4, 1993.
- [5] T. Breuel. Recognition of handprinted digits using optimal bounded error matching. In *Proceedings of the 2nd International Conference on Document Analysis and Recognition (ICDAR '93)*, page 493ff, 1993.
- [6] J. Canny. A computational approach to edge detection. In *IEEE Trans. Pattern Analysis and Machine Intelligence (8)*, page 679.
- [7] S. Chang. Extracting skeletons from distance maps, 2007.
- [8] X. Chen and A. Yuille. Detecting and reading text in natural scenes. In *Proceedings Computer Vision and Pattern Recognition (CVPR'04)*, pages 336–373, 2004.
- [9] B. T. Chun, Y. Bae, and T. J. Kim. Automatic text extraction in digital videos using fft and neural network. In *Proceedings of IEEE International Fuzzy Systems Conference Vol. 2*, pages 1112–1115, 1999.
- [10] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. In *Machine Learning (29)*, pages 103–130, 1997.
- [11] B. Fuglede and F. Topsøe. Jensen-shannon divergence and hilbert space embedding. In *Proceedings of the International Symposium on Information Theory*, pages 31–37, 2004.
- [12] R. C. Gonzalez and R. E. Woods. *Digital image processing*. Prentice Hall, 2002.
- [13] W. E. L. Grimson. *Object recognition by computer*. MIT Press, 1990.
- [14] Y. M. Hasan and L. Karam. Morphological text extraction. In *IEEE Transactions on Image Processing, 9 (11)*, pages 1978–1983, 2000.
- [15] B. Jähne. *Digital Image Processing*. Springer, Berlin, 1997.
- [16] A. K. Jain and B. Yu. Automatic text location in images and video frames. In *Pattern Recognition, 31 (12)*, pages 2055–2076, 1998.
- [17] K. Jung. Neural network-based text location in color images. In *Pattern Recognition Letters, 22 (14)*, pages 1503–1515, 2001.
- [18] K. Jung, K. I. Kim, and A. K. Jain. Text information extraction in images and video: a survey. In *Pattern Recognition, 37 (5)*, pages 977–997, 2004.
- [19] F. Kimura and M. Shridhar. Handwritten numerical recognition based on multiple algorithms. In *Pattern Recognition 24 (10)*, pages 969–983, 1991.

- [20] A. Kundu, Y. He, and P. Bahl. Recognition of handwritten word: first and second order hidden markov model based approach. In *Pattern Recognition 22 (3)*, page 283.
- [21] C. Lee and A. Kankanhalli. Automatic extraction of characters in complex images. In *International Journal of Pattern Recognition and Artificial Intelligence 9 (1)*, pages 67–82, 1995.
- [22] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, page 707.
- [23] H. Li, D. Doerman, and O. Kia. Automatic text detection and tracking in digital video. In *IEEE Transactions on Image Processing, 9 (1)*, pages 147–156, 2000.
- [24] J. Liang, D. Doermann, and H. Li. Camera-based analysis of text and documents: a survey. In *International Journal on Document Analysis and Recognition (7)*, pages 84–104, 2005.
- [25] Y. Liu and S. Srihari. Document image binarization based on texture features. In *IEEE Trans. Pattern Analysis and Machine Intelligence (19)*, pages 540–544, 1997.
- [26] S. M. Lucas. ICDAR 2005 text locating competition results. In *Proceedings Document Analysis and Recognition (1)*, pages 80–84, 2005.
- [27] S. M. Lucas, A. Panaretos, L. Sosa, A. Tang, and W. S. et al. ICDAR 2003 robust reading competitions: entries, results and future directions. In *International Journal on Document Analysis and Recognition - Special Issue on Camera-based Text and Document Recognition (7)*, pages 105–122, 2005.
- [28] A. Meijster, J. Roerdink, and W. Hesselink. A general algorithm for computing distance transforms in linear time. In *Mathematical morphology and its applications to image and signal processing*, 2000.
- [29] S. Messelodi and C. M. Modena. Automatic identification and skew estimation of text lines in real scene images. In *Pattern Recognition, 32*, pages 791–810.
- [30] G. K. Myers, R. C. Bolles, Q. T. Luong, J. A. Herson, and B. A. Hrishikesh. Rectification and recognition of text in 3-d scenes. In *Document Analysis and Recognition 7 (2-3)*, pages 147–158, 2004.
- [31] J. Ohya, S. A., and S. Akamatsu. Recognizing characters in scene images. In *IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (2)*, pages 214–224, 1994.
- [32] D. T. Øivind, A. K. Jain, and T. Taxt. Feature extraction methods for character recognition-a survey. In *Pattern Recognition 29 (4)*, pages 641–662, 1996.
- [33] A. F. R. Rahman and M. C. Fairhurst. Multiple classifier decision combination strategies for character recognition: A review. In *Document Analysis and Recognition*, pages 166–194, 2004.
- [34] S. Ramesh. A generalized character recognition algorithm: a graphical approach. In *Pattern Recognition 22 (4)*, page 347.
- [35] T. Sato, T. Kanade, E. K. Hughes, and M. A. Smith. Video ocr for digital news archive. In *Proceedings of IEEE Workshop on Content based Access of Image and Video Databases*, pages 52–60, 1998.
- [36] J. C. Shim, C. Dorai, and R. Bolle. Automatic text extraction from video for content-based annotation and retrieval. In *Proceedings of International Conference on Pattern Recognition, Vol. 1*, pages 618–620, 1998.

- [37] K. Shirai, M. Wakabayashi, M. Okamoto, and H. Yamamoto. A study for high performance character extraction from color scene images. In *Workshop on Document Analysis Systems DAS2008*, 2008.
- [38] M. Shridhar and A. Badreldin. Recognition of isolated and simply connected handwritten numerals. In *Pattern Recognition 19*, pages 1–12, 1986.
- [39] M. Smith and T. Kanade. Video skimming for quick browsing on audio and image characterization. In *Technical Report CMU-CS-95-186*, 1995.
- [40] I. Sobel and G. Feldman. A 3x3 isotropic gradient operator for image processing. unpublished but often cited, such as in *Pattern Classification and Scene Analysis* by Duda, R. and Hart, P. and Wiley J., 1968.
- [41] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, 2001.
- [42] I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.
- [43] V. Wu, R. Manmatha, and E. R. Riseman. Finding text in images. In *Proceedings of ACM International Conference on Digital Libraries*, pages 1–10, 1997.
- [44] V. Wu, R. Manmatha, and E. R. Riseman. Textfinder: An automatic system to detect and recognize text in images. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21 (11), pages 1224–1229, 1999.
- [45] J. Yang, J. Gao, J. Zhang, and W. A. Towards automatic sign translation. In *Proceedings Human Language Technology*, 2001.
- [46] J. Yang, J. Gao, J. Zhang, X. L. Chen, and W. A. An automatic sign recognition and translation system. In *Proceedings Workshop on Perceptive User Interfaces*, 2001.
- [47] J. Zhang, H. A., J. Yang, and A. Waibel. A robust approach for recognition of text embedded in natural scenes. In *Proceedings 16th International Conference on Pattern Recognition (ICPR'02)*, pages 204–207, 2002.
- [48] Y. Zhong, K. Kalle, and A. Jain. Locating text in complex color images. In *Pattern Recognition 28 (10)*, pages 1523–1535, 1995.