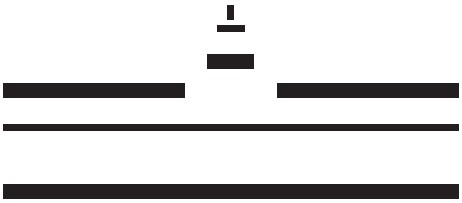

WESTFÄLISCHE
WILHELMS-UNIVERSITÄT
MÜNSTER

Barcode Detection and Recognition in Non-constrained Systems

Ausarbeitung

Veranstaltung:
Begleitendes Praktikum zu Computer Vision WS 2016/17

Themensteller:	**Prof. Dr. Xiaoyi Jiang**
	Dimitri Berh
	Andreas Nienkötter
Betreuer:	Aaron Scherzinger
Verfasser:	**Aleksej Matis**
	Alexander Schlüter
	Kjeld Schmidt

Barcode Detection and Recognition in Non-constrained Systems

Aleksej Matis, Alexander Schlüter, Kjeld Schmidt

24 March 2017

Contents

1 Introduction **3**

2 Localization **4**
2.1 Gradient+Blur 4
2.2 LSD 5

3 Boundary Detection **7**
3.1 Variation 7
3.2 LSD Bound 8
3.3 Wachenfeld 8

4 Reading **10**

5 Comparison **11**
5.1 Pros and Cons 11
5.2 Datasets 13
5.3 Run-Time Comparison 13

6 Improvements **14**
6.1 Gradient+Blur Detection 14
6.2 LSD 14
6.3 Wachenfeld 15
6.4 Template Matching 15

7 Conclusion **16**

1 Introduction

- Motivation
- Barcodes und EAN13
- Aufteilung in Lokalisation+Boundary+Reading

2 Localization

In order to read a barcode, we first have to localize it in a given image. We implemented two approaches. Gradient+Blur is a simple algorithm, with which we had some initial success. We later discarded it in favor of the much more successful Line Segment Detector approach. Both are detailed in this chapter.

2.1 Gradient+Blur

Our initial localization attempt showed limited success, but is conceptually very simple. The Gradient+Blur approach seeks to exploit the strong boundaries between the (white) background area and black bars in a standard barcode. In a few words; Detect edges on the image, then blur the result to remove noise and apply a threshold to the result. Open and close the resulting shapes, then select the biggest one. We will now go trough the steps in more detail.

1. Edge Detection

   We begin by detecting edges with the Sobel operator. We calculate the absolute gradient, discarding direction information. This leads to slightly worse results for barcodes rotated by about 45 degrees. The resulting image will usually contain large amounts of noise. This is fixed in the next step.

2. Blur+Threshold

   Next, we blur the previous result to smooth out the noise and apply a threshold afterwards. The blur is a normalized box filter of size 9. Note that 'noise' here refers not only to image imperfections, but also noise in the absolute gradient which might be a result of complex textures or other sources. The threshold is not set as a constant brightness of 127. Instead, we calculate the mean brightness of the image, add that value to 255 and divide by 2. This simple extra step gives slightly better results when dealing with very noisy images.

3. Close Forms

   The next two steps are very similar operations. First we *close* the remaining white areas. This means we first dilate the image, then erode it by the same amount. Dilation creates a new image in which a pixel is set to white if there is at least one white pixel within a given radius white in the original image. More

intuitively, white pixels grow outwards. Erosion is the opposite operation; A pixel is set to black, unless all pixels in the radius are white. This *closes* empty spaces between neighboring and inside of structures, while structures with no neighbors will remain unchanged. At this point, the barcode will ideally have merged into a solid shape,

4. Open Forms

   Opening the forms is similar to closing, but in reverse order; First eroding, then dilating by the same amount. This will eliminate small structures (which is not actually important at this point), while straightening edges on larger structures.

5. Selection

   Finally, we select the largest structure to be the most likely location of the barcode in the original image. In most cases, our result will be a jagged shape, only roughly tracing a rectangle. To simplify matters going forward, we now find the minimal area rectangle around the selected structure. This will get passed into the boundary detection step.

2.2 LSD

This localization method is based on the LineSegmentDetector algorithm [3], which is implemented in OpenCV 3 in the cv::LineSegmentDetector class [1]. The algorithm detects line segments in an image, such as the segments formed by the bars in a barcode. On a high level, it works by first calculating the image gradient and assigning to each pixel a unit vector perpendicular to the gradient, resulting in a *level line field*. Pixels with a gradient magnitude under a certain tolerance are discarded. Connected pixels with similar level line angles are clustered together and form a *line support region*. To each line support region, a rectangle is fit which covers the whole region. To limit the number of false detections, the rectangle is only accepted as a line segment if the number of covered pixels with aligned level lines is high enough relative to the total number of covered pixels. The exact threshold is based on a statistical model (*a contrario* method), so that the rectangle is only accepted if, in a purely random level line field, the found ratio of aligned level lines is unlikely. Before a rectangle is rejected, some variations of the rectangle's parameters are tried.

The barcode localization method is taken from Creusot et al. with minor modifications. First we run the LineSegmentDetector provided by OpenCV on the image. We

use default parameters, except for the first parameter LSD_REFINE_NONE, which disables refinement of line segments. We found that line refinement tends to break up barcode bars into multiple segments, e.g. if the bars in the image are not quite straight due to kinks in the material on which the barcode is printed, or if there are glare spots on the barcode. This is undesirable, since in the following steps we rely on the length of line segments in the barcode to be similar.

Next we want to find a line segment which belongs to a barcode bar approximately in the middle of the barcode. To each detected line segment, a score is assigned based on how many other line segments might belong to bars of the same barcode as the first one. Two line segments are regarded as possibly belonging to the same barcode, if

1. their centers are not too far apart,
2. they have similar length,
3. they have similar angles and
4. their projected intersection covers most of the smaller segment.

Calculation of the first three distance measures is straightforward: Let c_i be the center positions, l_i the segment lengths and α_i the segment angles, where the index $i = 1, 2$ denotes the first or the second line segment, respectively. The used criteria are

$$d_{\text{center}} \coloneqq \frac{\|c_1 - c_2\|}{l_1} < 1, \qquad \frac{|l_1 - l_2|}{l_1} < 0.3, \qquad |\alpha_1 - \alpha_2| < 0.1\,.$$

As BIIIIIIILD shows, two lines fulfilling the first three criteria might still not be arranged like barcode bars, if they are displaced along the line segment direction. To catch these cases, we project the second line onto the first and check that the intersection is large enough relative to the length of the smaller line, see BIIIIIIILD2.

For each line segment which fulfills the criteria, Creusot et al. increment the score of the first line by one. Instead, we increment the score of the first line segment by $1 - d_{\text{center}}$. This effectively weights the score by the distance between the line segments, which favors line segments which lie in the middle of the barcode.

Finally, we select the line segment with the highest score. This line segment should correspond to a barcode bar in the middle of the barcode.

3 Boundary Detection

To complement the LSD localizer, which returns a line segment in the middle of the barcode (hereafter called *best line*), we need to find the barcode's boundaries before we can attempt to read it.

3.1 Variation

The variation boundary finder is the original boundary detection method proposed in [2] in conjunction with the LSD localization. Let $L_\perp$ denote the sequence of intensity values along the line perpendicular to the best line, going through the best line's center (hereafter called *bisector*). Since barcodes consist of alternating black and white bars, the variation of intensity values along a line crossing the barcode is expected to be high. The boundary points are thus expected to have the following property: Extend a "probing" line (*inner probe*) from the boundary point along the bisector, pointing inside the barcode, i.e. in direction of best line's center. Extend a second "probing" line (*outer probe*) from the boundary point in the opposite direction, away from the barcode. The variation along the inner probe is now expected to be much higher than the variation along the outer probe.

To find the boundary points, for each index k along the bisector, the signed variation difference

$$\phi_{L_\perp}(k) = \sum_{j=-R}^{0} |L_\perp(k+j) - L_\perp(k+j-1)| - \sum_{j=0}^{R} |L_\perp(k+j) - L_\perp(k+j+1)|$$

is computed. Here, R is the probing distance, which we chose as $R = l_*/2$, half of the length of the best line.

The right boundary point should lie at the bisector index with maximal $\phi_{L_\perp}$, while the left boundary point should have minimal $\phi_{L_\perp}$.

We deviate slightly from Creusot et al. by instead using the variation measure

$$\phi^*_{L_\perp}(k) = \sum_{j=-R}^{0} |L_\perp(k+j) - L_\perp(k+j-1)| \cdot (R+j) - \sum_{j=0}^{R} |L_\perp(k+j) - L_\perp(k+j+1)| \cdot (R-j)\,,$$

giving higher weight to pixel values near the current boundary candidate at index k.

3.2 LSD Bound

Motivated by the high quality of line segments returned by LineSegmentDetector, we constructed a new boundary finder, using ideas from the variation method described above. The goal is to determine the line segments corresponding to the two outer barcode bars, the *boundary segments*.

We first filter the line segments, so that only segments parallel to and located next to the best line are kept. To that end, we use the angle criterion and the criterion of sufficient projected intersection as described in section 2.2. The remaining segments are sorted based on the position of the projection of their centers onto the bisector line. The boundary segments should have the following property: As before, extend an inner probe from the projected center of the boundary segment along the bisector, pointing inside the barcode. Extend an outer probe in the opposite direction, away from the barcode. The inner probe is expected to cross a large number of line segments parallel to and located next to the best line (i.e. segments which have not been filtered out in the previous step). The outer probe is expected to cross fewer such lines. ...

3.3 Wachenfeld

Another implementation was adapted from the barcode detection and recognition algorithms proposed by Wachenfeld et al. [4]. The method is subdivided into the three steps of preprocessing, barcode edge detection and hypothesis based reading. Since we implemented barcode localization (section 2) and barcode reading (section 4) as separate steps only the edge detection with a bit of preprocessing is adapted.

The originally proposed boundary detection assumes that the position of the barcode is centered. This way a simple scan line can be applied to the picture. If the barcode is unreadable or misaligned the scan line can be reapplied with a rotation or shift factor. Since the position and rotation of the barcode is already determined in the localization step it is sufficient to place a scan line perpendicular to the best line. To simplify further steps we applying a affine transformation to the region around the scan line.

For the preprocessing we need to first detect to local maxima and minima. Our pictures are already converted to gray scale on loading so we can directly traverse the scan line from to middle pixel toward the outside and look for neighboring pixels with difference in value greater then a set threshold. In a second step we need to remove overly extreme points by executing a pruning step. Lastly we need to apply a dynamic

Figure 1: Application of the Wachenfeld boundary detection. The selection is colored in red. Steps are shown from top to bottom.

threshold by always comparing every local extrema to an inwards neighborhood.

The boundary detection is based on some properties of the barcode. As shown in section 1 a EAN13 barcode consist of 95 alternating black and white lines. These lines can only be arranged based on the 20 patterns defined by the specification. The patterns also have the property that every patter has exactly 4 alternations. This means a full barcode has 59 segments of black and white: 2×3 for the outer guard, 5 for the inner guard and 12×6 for the 12 digits.

To detect the boundaries we start with the middle pixel. If this pixel is white we look for the nearest black pixel. The corresponding black segment of this pixel is selected. From this selection we traverse toward the outsides by looking for the smallest neighboring pair of white and black segments to the left and right and adding it to the current selection. The selected region must be at all times be enclosed by black segments. This process is demonstrated in fig. 1. The smallest pair of segments is required to prevent the algorithm from selecting the quit zone.

The outer guard bars are detected as soon as we have selected all 59 black and white segments respectively 29 pairs and the initial middle segment.

4 Reading

Hier nur Template Matching

5 Comparison

5.1 Pros and Cons

Gradient + Blur

Pros:

Gradien+Blur is conceptually extremely simple and therefor extremely fast. It also is robust against disturbances like reflections since it only looks for the biggest contour.

Cons:

Since the algorithm uses the absolute gradient it is quite weak against rotations of 45 degrees. Gradient+Blur is also dependent on many parameters. If these parameters don't match up with the barcode the algorithm fails. One such instance is a too small kernel size for the closing step, which leads to a fragmented contour because it fails to close the white segments. Another problem is the susceptibility to noise. Gradient+Blur could mistake the wrong contour for the barcode.

LSD

Pros:

The LSD is still relatively fast and can reliably detect a barcode in almost every position and rotation.

Cons:

The LSD looks for all parallel lines. If other parallel lines are parallel to the barcode and close enough they can be added to the barcode by mistake. Disturbances like reflections can also break barcode lines and lead to exclusion valid lines.

Variation

Pros:

Variation

Cons:

Variation

LSD Bound

Pros:

The LSD Boundary Detection is extremely accurate since it test all significant lines determent by the LSD localization and returns multiple potentiality boundary points.

Cons:

Since it is strongly dependent on the LSD localization it fails as soon as the LSD localization fails. And since an enormous amount of lines needs to be tested the algorithms is not suited for real time applications.

Wachenfeld

Pros:

The Wachenfeld algorithm is relatively quick since it is based on a scan line.

Cons:

The algorithm is strongly dependent on the quality of the preprocessing and detection of local extrema. And since the algorithm is based on a scan line it has no way to deal with broken barcodes caused by disturbances. Finally if the last segment selection steps misses the barcode can be shifted to the left or right and become unreadable.

Template Matching

Pros:

Since no binarization is required template matching is extremely robust against bluered barcodes. The algorithm also uses percentages to read the barcode and is therefor flexible when encountering errors.

Cons:

The matching process required precomputed data to be matched against the barcode. This can take a lot of time and memory depending on the targeted accuracy. And since the template matching is based on probability it is more likely to return a false positive result than no result.

	Errors	Hit rate	Time in sec.
Gradient + Blur	831	22%	266
LSD + Wachenfeld	452	56%	319
LSD + LSDBounds	41	97%	1760
LSD + Variation	279	74%	1595

Figure 2: Comparison of different algorithms on the WWU Muenster Barcode Database.

5.2 Datasets

The implemented algorithms were tested on three datasets.

- Generated Barcodes:
  - 110 images, 300×150 px, barcode only.
  - 110 images, 1000×1000 px, barcodes (300×150 px) randomly rotated and translated on white background.
- WWU Muenster Barcode Database [5][4]:
  1055 images, 800×600 px.
- ArteLab Dataset - Robust Angle Invariant 1D Barcode Detection [6] [8] [7]:
  2 sets of 215 images, 800×600 px.

5.3 Run-Time Comparison

The algorithms were compared over the 1055 images of the WWU Muenster Barcode Database [5]. The programm was implemented in C++ with the help of the Qt 5.7 and OpenCV 3.1 frameworks. The test were made on a Debian 8 system with an Intel N2940 CPU, 4 Cores / 4 Threads, and 4 GB RAM. The results are shown in fig. 2.

6 Improvements

In this section, we will consider some possible improvements of our algorithms.

6.1 Gradient+Blur Detection

Vary by rotation

Since the Gradient+Blur-detection builds from the absolute gradients found by the Sobel operator, results might be poor when a barcode is rotated close to 45 degrees. To remedy this, we might run the detection on rotated versions of the image. Rotating once by 45 degrees would yield the greatest improvement, however, multiple rotations could be checked at diminishing returns.

Vary kernel sizes

In our current implementation, all operations (blurring, closing, opening) are applied with a constant kernel size. This is somewhat problematic, since the effectiveness of these parameters is dependent on the size of the barcode. If a barcode is not detected successfully, we might try different kernel sizes.
The initial kernel size might also be considered further: The current values have been empirically determined to be most useful for the image sizes we worked with. Scaling the kernel size appropriately for each image might lead to better results, since it seems likely that the relative sizes barcodes in images is independent of image resolution.

6.2 LSD

Unite interrupted line segments

A remaining common failure of the LSD algorithm is it's poor handling of reflections on the barcode. These reflections visually interrupt the barcode lines, so two short line segments are returned instead of one. This is problematic, as length of line segments is an important factor in deciding whether two line segments might be part of the same barcode. This ultimately leads to a split in the barcode, where the detected area covers just the larger side of the code.

This problem could be fixed in two related ways: First, we could try a *line segment unification*. This process would seek to unite line segments which lie on the same line

in space. This easy calculation would unite line segments across gaps, thus restoring the proper length and avoiding rejection. To avoid uniting many stray lines across the image, simply limit unification to line segments where the gap size is less than some small multiple of the segment size itself.

This first approach has the weakness of only working when the reflection splits the line segment into two parts. Even better would be an approach that could handle a missing endpoint. This should be handled by this second approach, which is also more complicated: Instead of uniting line segments, we would now consider the endpoints of line segments in a second pass over all segments. During the first pass, whenever a pair of lines is considered to likely be part of the same barcode, save the location of their endpoints in two different lists, based on distance from the origin. After this first pass, those lists can easily construct the upper and lower bound of the barcode. With this new information, discard the previous scores and redo them. However, this time, instead of checking for similar length, check whether two line segments have an endpoint near one of those two lines before assigning a score to them. This second pass will now also handle missing endpoints.

6.3 Wachenfeld

Proper preprocessing, multiple scanlines

Since we have found quick success with the LSD-Boundary detection, our current implementation of the Wachenfeld algorithm does not prepare the image in the way recommended in the original paper, that is, there is no adaptive thresholding along the scanline, which is a major factor of success for the original implementation. Thus, proper initialization would likely greatly improve our results. Also, we could detect along multiple scanlines and vote for most likely boundaries.

6.4 Template Matching

Precompute patterns for blur of various strengths

While the algorithm is already very capable in handling blurred barcodes, this might be improved even further by precomputing patterns specifically not for an ideal, clean barcode, but for one that is already blurred.

7 Conclusion

References

[1] G. Bradski. *cv::LineSegmentDetector Class Reference*. Mar. 20, 2017. URL: http://docs.opencv.org/3.1.0/db/d73/classcv_1_1LineSegmentDetector.html.

[2] Clement Creusot and Asim Munawar. "Low-computation egocentric barcode detector for the blind". In: *Image Processing (ICIP), 2016 IEEE International Conference on*. IEEE. 2016, pp. 2856–2860.

[3] Rafael Grompone von Gioi et al. "LSD: a Line Segment Detector". In: *Image Processing On Line* 2 (Mar. 2012), pp. 35–55. DOI: 10.5201/ipol.2012.gjmr-lsd.

[4] Steffen Wachenfeld, Sebastian Terlunen, and Xiaoyi Jiang. "Robust recognition of 1-d barcodes using camera phones". In: *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*. IEEE. 2008, pp. 1–4.

[5] Steffen Wachenfeld, Sebastian Terlunen, and Xiaoyi Jiang. *WWU Muenster Barcode Database*. URL: http://cvpr.uni-muenster.de/research/barcode.

[6] Alessandro Zamberletti, Ignazio Gallo, and Simone Albertini. *Robust Angle Invariant 1D Barcode Detection*. Proceedings of the 2nd Asian Conference on Pattern Recognition (ACPR), Okinawa, Japan. 2013. URL: http://artelab.dista.uninsubria.it/downloads/datasets/barcode/hough_barcode_1d/hough_barcode_1d.html.

[7] Alessandro Zamberletti, Ignazio Gallo, and Simone Albertini. "Robust angle invariant 1d barcode detection". In: *Pattern Recognition (ACPR), 2013 2nd IAPR Asian Conference on*. IEEE. 2013, pp. 160–164.

[8] Alessandro Zamberletti et al. "Neural Image Restoration for Decoding 1-D Barcodes using Common Camera Phones." In: *VISAPP (1)*. 2010, pp. 5–11.