BARCODE DETECTION WITH MORPHOLOGICAL OPERATIONS AND CLUSTERING

Péter Bodnár
Department of Image Processing
and Computer Graphics
University of Szeged
Árpád tér 2., Szeged, H-6720 Hungary
email: bodnaar@inf.u-szeged.hu

László G. Nyúl
Department of Image Processing
and Computer Graphics
University of Szeged
Árpád tér 2., Szeged, H-6720 Hungary
email: nyul@inf.u-szeged.hu

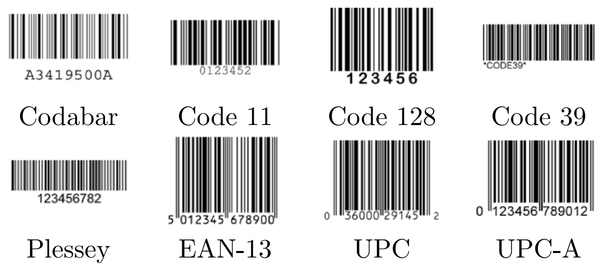


Figure 1: Barcode patterns

ABSTRACT
Barcode detection has many applications and detection methods. Each application has its own requirements for speed and detection accuracy. Fine-tuning, upgrading or combining existing methods gives fast and robust solutions for detection. Modern computer vision techniques help the whole process to be fully automated. Different detection approaches are examined in this paper, and new methods are introduced.

KEY WORDS
barcode detection, computer vision, clustering, feature extraction

1 Introduction

Barcodes are 1D codes that consist of a well-defined group of parallel lines aiming easy automatic identification of carried data with endpoint devices such as PoS terminals, smartphones, or computers. Barcode decoding is fast and most barcode standards provide redundant information for error correction purposes. 2D codes are also referred to as barcodes, but in this paper we restrict ourselves only to codes like those showed in Fig. 1.

Barcode detection methods have two main objectives, speed and accuracy. On smartphones, fast detection of barcodes is desireable, but accuracy is not so critical since the user can easily reposition the camera and repeat the “scan”. Accuracy, however, is critical for industrial applications (e.g. postal services), where false negatives cause loss of profit. Speed is also a secondary desired property in these applications.

The basic approach for barcode detection is scanning only one, or just a couple of lines of the whole image. This method is common at hand-held PoS laser scanners or smartphone applications. Scanned lines form an 1D intensity profile, and barcode-detector algorithms [1, 2] work on these profiles to find an ideal binary function that represents the original encoded data. The advantage of this technique is time efficiency and low hardware requirements. In most cases, only a subset of image pixels are read, and a small buffer capable of storing a few image lines is sufficient. However, that approach often requires multiple scans of the same scenario due to the lower accuracy.

Another approach is to extract texture-like properties and detect properties that refer to barcode-like appearance. These methods use clustering and morphological operations [3]. In this paper we experiment with the latter class of techniques with different parameters and also in combination with other approaches.

2 The proposed barcode detection methods

In this section we introduce two new methods, both approaches are based on texture analysis, one globally while the other both locally and globally. They are compared with standard feature detection.

2.1 Preprocessing

The digital image acquired from the camera often needs preprocessing because of device flaws or environmental difficulties. On images having low contrast, intensity levels should be normalized. We also use unsharp masking, which is the weighted addition of the original image pixel intensities to the negated pixel values of the gaussian-blurred version of the image. The blurring gaussian filter is adjusted to not to destroy the narrowest line of the barcode. Since one of the proposed methods works on binary images, thresholding is necessary. A simple threshold is sufficient on images with even lighting, otherwise adaptive thresholding [4] is required.

Image resolution does not have to be high. Barcodes having the narrowest line of two pixels is sufficient and 3×3 px median filters can be applied to eliminate salt-and-pepper noise. Higher resolution produces better results, but also increases computation time. The least time-consuming solution for downsampling such images is the nearest neighbour interpolation, which is also a good choice because it preserves hard edges, that constitute a desired pattern of a barcode. Since the test suite contains synthetic images with barcodes having 2 pixels as the narrowest line, downsampling was not necessary. In our real-life examples, every code has at least 2 px to 4 px minimum line width. Images with larger barcode resolution were downsampled before rendering them to abstract background images.

2.2 The Canny + Hough method

This method is not considered as a specific barcode detection method, it is presented here as a generic reference. It only applies general image processing methods like Canny [5] edge detection and Probabilistic Hough transform [6], as barcodes consist of roughly equally long, parallel lines in a small area. It gives a probabilistic estimation for detecting straight lines with the help of a subset of the edge points of the original image, outperforming the standard Hough [7] transform. For preprocessing, we use a blur filter since smooth images are desired for Canny edge detector. Since all barcodes in the test suite have at least 64 px bar height, we set the minimum line length to 50 in the Hough transformation.

After we obtain a list of lines with their center point, length, and orientation, we can cluster them to decide wheather they constitue a barcode or not. We define the minimum number of lines, the proximity needed for the lines to be in the same cluster, and the tolerance for length and orientation from the means inside the cluster. Since our barcodes consist of at least 25 parallel lines, we defined the minimum number of lines as 20. In the final step, cluster centers are returned, and the image can be cropped for decoding with known barcode decoding implementations (Fig. 2).

2.3 MIN–MAX operations

This method treats the image as a whole, and therefore requires a fair amount of RAM and computation time. Supposed that intensity levels have been normalized before, no other preprocessing operations

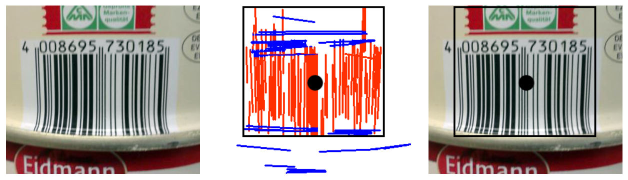

(a) original image (b) feature image (c) overlay

Figure 2: Canny edge detector with Probabilistic Hough transform. In (b), detected lines that are part of a barcode-like cluster are shown in red while the other detected lines are shown in blue

are required since this method manages well noisy, blurry or distorted images. Knowing the maximum bar width of a barcode, we apply the morphologic gradient (`dilate()` − `erode()`) operator on the image with a box kernel of size $2 \times \lfloor \texttt{max\_barwidth}/2 \rfloor + 1$. The next step is removing ghost elements (see Fig. 3b) from the feature image with a binary threshold. A good threshold can be at 75 % of the full intensity scale (e.g. 192 for 8-bit grayscale images), since barcodes produce areas close to the maximum intensity. After that, we apply morphologic opening operation (`dilate(erode())`) on the feature image, with the previously defined kernel for closing the small gaps caused by scratches, reflections or other flaws of the original image.

At this stage we already have a feature image showing the barcode-like areas with white, the last thing we have to do is to compute the exact area of these areas, and their momentum. Experiments showed that setting the minimum area to be classified as barcode to $w \times h \times 0.75$ or lower is satisfactory (where w and h are barcode width and height respectively).

2.4 Local clustering

This method came from examining the behaviour of textures. Texture parts have similar local statistics in their neighbourhood, so dividing the image to square tiles and examining each tile locally and also globally makes the base of the algorithm. We obtained the appropriate size for tiles by experimenting. The local statistics of each tile is examined and then every tile makes one value of a global feature matrix that shows us possible barcode areas.

The main idea of Local clustering is that an image region that contains a barcode segment has many similar stretched pixel clusters (Fig. 4). The minimum count of expected clusters can be derived from the widest bar of the barcode. Degree of stretch can be measured with the diameter of the cluster (defined as twice the distance of the furthest cluster point from the cluster center). With exactly horizontal or verti-

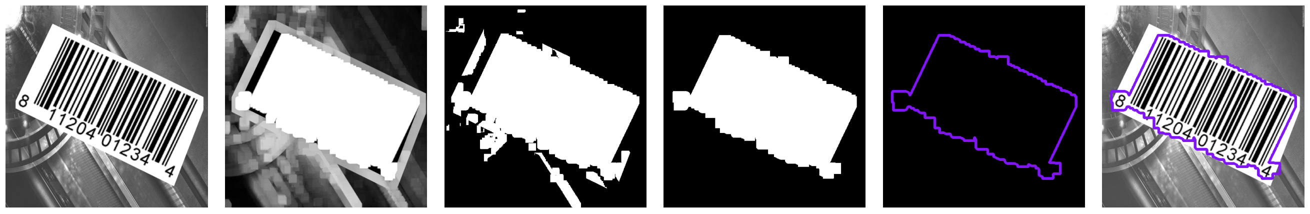


Figure 3: Stages of MIN–MAX method

cal lines, the largest cluster diameter is the tile size, in oblique situations, the largest cluster diameter is expected to be longer than that. Furthermore, stretched separate clusters need to be aligned approximately identically, otherwise one cluster would touch another, decreasing the number of separate clusters in a tile below our threshold. For preprocessing, we use median filter first that eliminates salt-and-pepper noise. On real-life images having low contrast at barcode areas, adaptive thresholding is necessary.

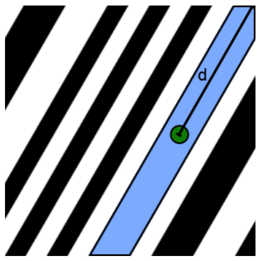


Figure 4: The idea of Local clustering. Here d is the maximum distance from the cluster center, i.e. the half of the cluster diameter

Another important property is the minimum cluster size in pixels. This can be easily computed from `min_barwidth` $\times$ `tile_size`.

After evaluating all tiles locally, we look for clusters in the feature matrix. This matrix has compact areas with high values where barcodes can be found in the image (Fig. 5). In our algorithm we applied a threshold for values to classify whether or not an area contains barcode segment. Tests showed that a choice of 0.5 or above for this parameter is satisfactory. Defining the threshold above this value decreases detection accuracy, while setting below 0.5 increases false positive rate significantly.

This binarized feature matrix can be clustered via connected component labeling [8]. Finally, small components are dropped, and momentums of the remaining clusters are returned. Clusters are considered small when they contain less than N tiles (Eq. (1)), where s is the tile size (which is 1/3 of the code height in our examples). Bounding boxes in our examples are not enclosing the whole barcode in every case. This is because we only calculated the lower and upper bounds for the clusters in the feature matrix, and barcode corner pieces are too weak for the feature. The bounding boxes can be simply improved by finding aligned rectangles instead.

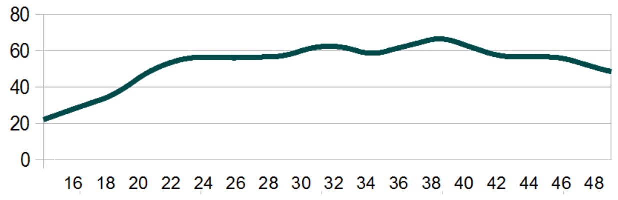


Figure 6: Detection accuracy with respect to the tile size. X-axis: proportion of tile size and barcode height; Y-axis: detection accuracy (both expressed in percent).

$$N = \max\left(2, \frac{|h - s| \times |w - s|}{s^2}\right) \qquad (1)$$

Since the smallest barcode in our set has a 60 px height, 30×30 px or greater tile sizes have poor recognition capability. However, very small tile sizes also lead to greater error for computing the center of the codes, because of the characters appearing below the code with code pieces nearby also have a barcode-like property (plain text is not affected). Also, choosing the tile size below two times the width of the widest barcode line leads to poor accuracy, since only two clusters can be detected on the tile, and that does not characterize a barcode part well. The best tiling size appears to be about 1/3 of the barcode height (Fig. 6). Since all examined codes consist of the same pattern (parallel lines), we looked for the optimal tile size for all types of codes together.

Running the method on the same scenario with different offsets yield different detection accuracy (Fig. 8c), which shows that this approach is sensitive to the choice of tiling. Further investigations are underway as to how select the best tiling offset, as well as possible setups with overlapping tiles (which obviously increases computational requirements). This 2-phase approach works as follows. In the 1st phase, Local clustering is performed with zero-offset tiling, and in a 2nd phase the same is done using an offset of half the

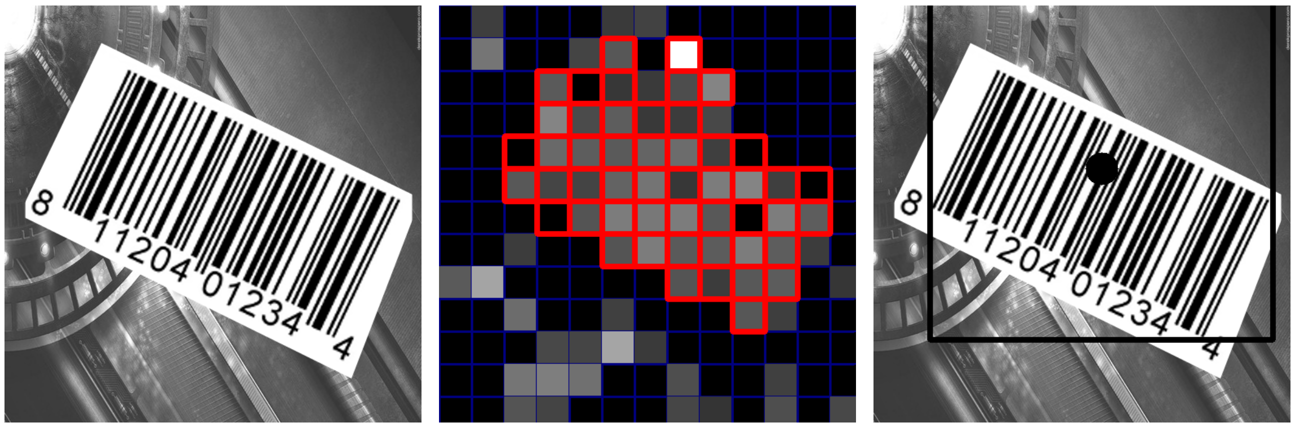

(a) original image (b) feature matrix visualized as gray values. Red squares are above threshold. (c) code center and bounding box

Figure 5: Stages of Local clustering

tile size in both directions. The code centers detected by the 2 phases are pooled together with an extra filtering wherein those code centers that are detected in both phases and are close to each other are merged into one (the larger cluster is kept), because they are likely to correspond to the same code.

3 Evaluation

The discussed methods were tested on a fair amount of images of different types of barcodes (Fig. 1), both synthetic and real.

3.1 Test Suite

We generated barcodes digitally with the types shown in Fig. 1. Pixel dimensions on Table 1, are measured without the numeric representation of encoded data. Every barcode has a minimum bar width of 2 px. Only one base image has been chosen for each code type, and it was sufficient because the chosen encoded data (as many numeric and alphanumeric characters as possible) represented well the various bar configurations.

We generated every combination on the base image set of the following properties: rotation in every $15°$ from $0°$ to $180°$, Gaussian blur filter with 5×5 kernel with 5 different σ (and without smoothing), additive noise from 0% to 50% with a step of 5%. Test set contained 8 different barcodes with 12 orientations, 6 different blur filters and 11 different rates of additive noise, with a total of 6336 images.

For more realistic examples, we googled up images of barcodes that had no distortion, good contrast and minor or no reflections. We also googled up abstract wallpapers and made 500 randomly cropped grayscale images from them. We embedded the cropped barcodes within the background images using randomly selected rotations from the 0–30° angular range around X, Y and Z axes, allowing for affine distortions.

Another 100 images containing barcodes were collected from real-life examples without any modifications to the images. Minor reflections, blur, scratches and distortions were present in these images. This set serves qualitative purposes only, and due to having low amount of images compared to the synthetic set, we do not manage it separately for test results.

3.2 Implementation and test environment

We implemented the method in `C++`, with the help of the `OpenCV` library. `C++` provides convenient OOP approach and fast code execution, while `OpenCV` has all the functions needed for image preprocessing and manipulation. Most operations were made by built-in functions, like finding the contours of a point set, measuring the area of a polygon and each morphological operations. Evaluation is performed on a computer with Intel(R) Core(TM)2 Duo 3.00GHz CPU.

3.3 Accuracy and detection speed

The Canny + Hough method is slow and resource-demanding, and only used here as a reference for comparison with other faster, more barcode-oriented detection methods. However, despite the slow speed, it is highly effective for finding barcode lines.

The MIN–MAX method is very tolerant to blur and noise. Stronger blur can be compensated with adding noise (Fig. 7a), as MIN–MAX produces more compact feature areas. This method is accurate, but convolutions are more time-consuming than simpler methods, like scanline analysis, and also requires more memory. For fine-tuning the parameters, we ran experiments with different kernel sizes for morphologic operations and different thresholds for area of accepted clusters (Fig. 7b).

Regarding efficiency, the MIN–MAX method takes approximately 420 ms per test image (512×512 px), while the Local clustering approach analyzes one image under 40 ms, and an additional 15-20 ms is required for unsharp masking. 2-phase local clustering takes about 100 ms, including unsharp masking.

Local clustering needs to read every pixel of an image, and calculate cluster centers and distances, which takes a fair amount of computation time, but is still considered a good compromise between runtime and efficiency. Local clustering is far more sensitive to noise and blur than MIN–MAX (Fig. 8a), because clusters on local tiles are easily separated by noise, or unified by heavy blur. Applying unsharp masking increases detection accuracy significantly (Fig. 8b). Furthermore, repeating the procedure with a tiling using a different origin, can change the accuracy, because flaws on image are less interfering at the edge than the center of a tile (Fig. 8c), thus careful choice of the tile size as well as the tiling origin is required for a robust setup.

Both MIN–MAX and Local clustering are tolerant to distorsions, like twist and ripples. MIN–MAX examines the image globally for texture-like behaviour ignoring distorsions, and Local clustering also performs well because local features bearly change this way. Both methods are tolerant to flaws on the barcode like text, as long as font weight is comparable to line width of the barcode. Greater amount of damage on the barcodes (e.g. wide brush strokes) breaks the Local clustering method first (Fig. 9) while the MIN–MAX method seems to give reasonable detection even under such distortion.

At 2-phase approach, selecting the best tiling origin for each particular image via exhaustive search would extremely slow down the method. However, by just two runs, on the expense of roughly doubling computation time, approximatly 10% accuracy improvement can be reached (Fig. 8c).

Test results (Table 1) show that detection statistics varies for different code types. These accuracy bounds depend on width-to-height ratio of the codes, besides the amount of noise and blur. For accuracy formula, Jaccard index can be used on the pixels of the original and detected bounding boxes (Eq. 2)

$$T_s(O, D) = \frac{\sum_{x,y}(O(x, y) \wedge (D(x, y))}{\sum_{x,y}(O(x, y) \vee (D(x, y))} \qquad (2)$$

where O and D are binary functions giving 1 on the inside of the original and detected bounding boxes respectively.

4 Concluding remarks

We have presented two novel approaches for detecting barcode regions using texture analysis, and studied their behavior on a set of images showing various barcodes. These methods are highly efficient and for certain types of codes also show high accuracy. We are also studying efficient barcode detectors using scanline analysis, as well as machine learning techniques. We expect that an ensemble of detectors specially deviced for certain code types can significantly improve the overall accuracy. In industrial setups parallel execution may be possible and then the ensemble efficiency remains comparable to that of a single detector.

References

[1] Timothy R. Tuinstra. *Reading Barcodes from Digital Imagery.* PhD thesis, Cedarville University, 2006.

[2] Eugene Joseph and Theo Pavlidis. Bar code waveform recognition using peak locations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(6):630–640, jun 1994.

[3] Daw-Tung Lin, Min-Chueh Lin, and Kai-Yung Huang. Real-time automatic recognition of omnidirectional multiple barcodes and dsp implementation. *Machine Vision and Applications*, 22:409–419, 2011. 10.1007/s00138-010-0299-3.

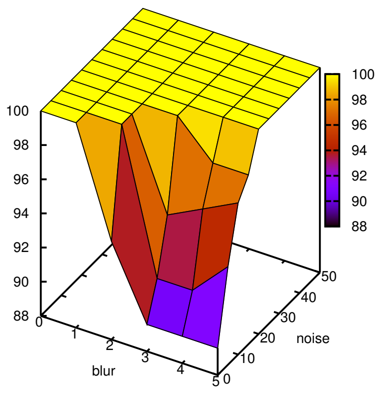
[4] Sue Wu and Adnan Amin. Automatic thresholding of gray-level using multistage approach. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 493–497 vol.1, aug. 2003.

[5] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698, nov. 1986.

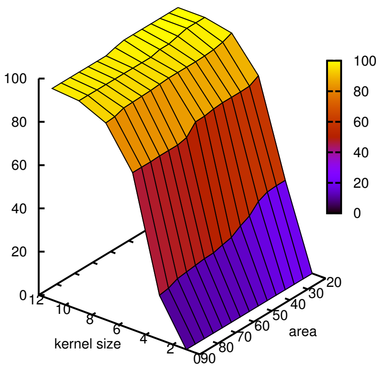
[6] Nahum Kiryati, Yuval Eldar, and Alfred M. Bruckstein. A probabilistic hough transform. *Pattern Recognition*, 24(4):303–316, 1991.

[7] Sherin M. Youssef and Rana M. Salem. Automated barcode recognition for smart identification and inspection automation. *Expert Systems with Applications*, 33(4):968–977, 2007.

[8] Michael B. Dillencourt, Hannan Samet, and Markku Tamminen. A general approach to connected-component labeling for arbitrary image representations. *J. ACM*, 39:253–280, April 1992.

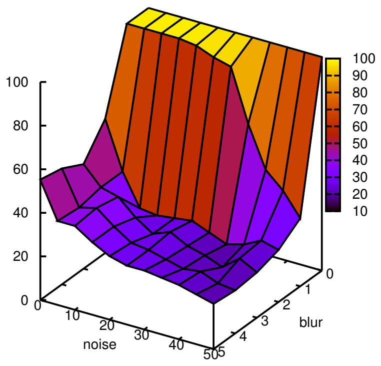


(a) accuracy with respect to noise and blur

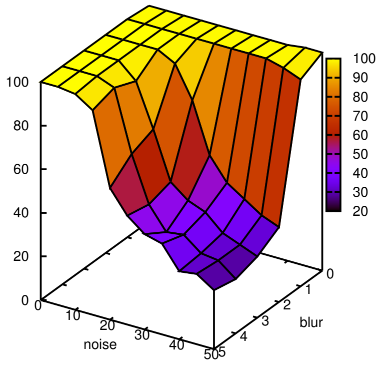


(b) accuracy with respect to kernel size and area threshold

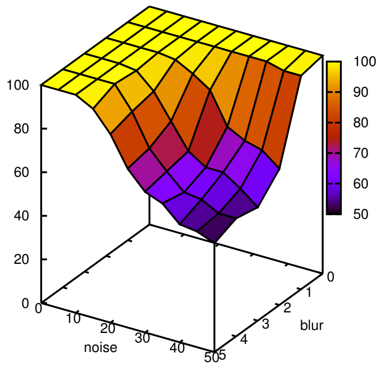
Figure 7: Detection accuracy of MIN–MAX



(a) without preprocessing



(b) with unsharp masking



(c) 2-phase local clustering

Figure 8: Detection accuracy of Local clustering

Table 1: Accuracy of MIN–MAX and Local clustering without and with unsharp masking (mean ± sd, expressed in percent)

Type	Size	Canny + Hough transform.	MIN–MAX	Local clustering without u. masking	Local clustering with u. masking	Local clustering 2-phase + u. masking
Codabar	188 × 100	96.7 ± 17.1	95.6 ± 19.7	29.7 ± 45.7	65.2 ± 47.7	72.0 ± 45.0
Code 11	176 × 64	98.7 ± 11.7	100.0 ± 0.0	69.4 ± 46.1	97.6 ± 15.3	99.6 ± 6.1
Code 128	180 × 150	97.2 ± 16.4	100.0 ± 0.0	44.6 ± 49.7	96.0 ± 19.7	99.7 ± 5.0
Code 39	332 × 100	60.7 ± 48.8	100.0 ± 0.0	29.0 ± 45.4	63.4 ± 48.2	78.8 ± 40.9
EAN-13	190 × 138	98.5 ± 12.2	100.0 ± 0.0	36.7 ± 48.2	69.7 ± 46.0	93.3 ± 25.0
Plessey	402 × 80	76.1 ± 42.2	94.7 ± 22.4	25.0 ± 43.3	58.8 ± 49.2	66.2 ± 47.3
UPC	187 × 96	99.7 ± 5.0	100.0 ± 0.0	56.0 ± 49.7	84.2 ± 36.5	92.6 ± 26.2
UPC-A	190 × 120	98.7 ± 11.1	100.0 ± 0.0	41.0 ± 49.2	75.5 ± 43.0	92.3 ± 26.7
All together		90.8 ± 20.6	98.8 ± 2.8	41.4 ± 49.3	76.3 ± 42.5	86.8 ± 33.9

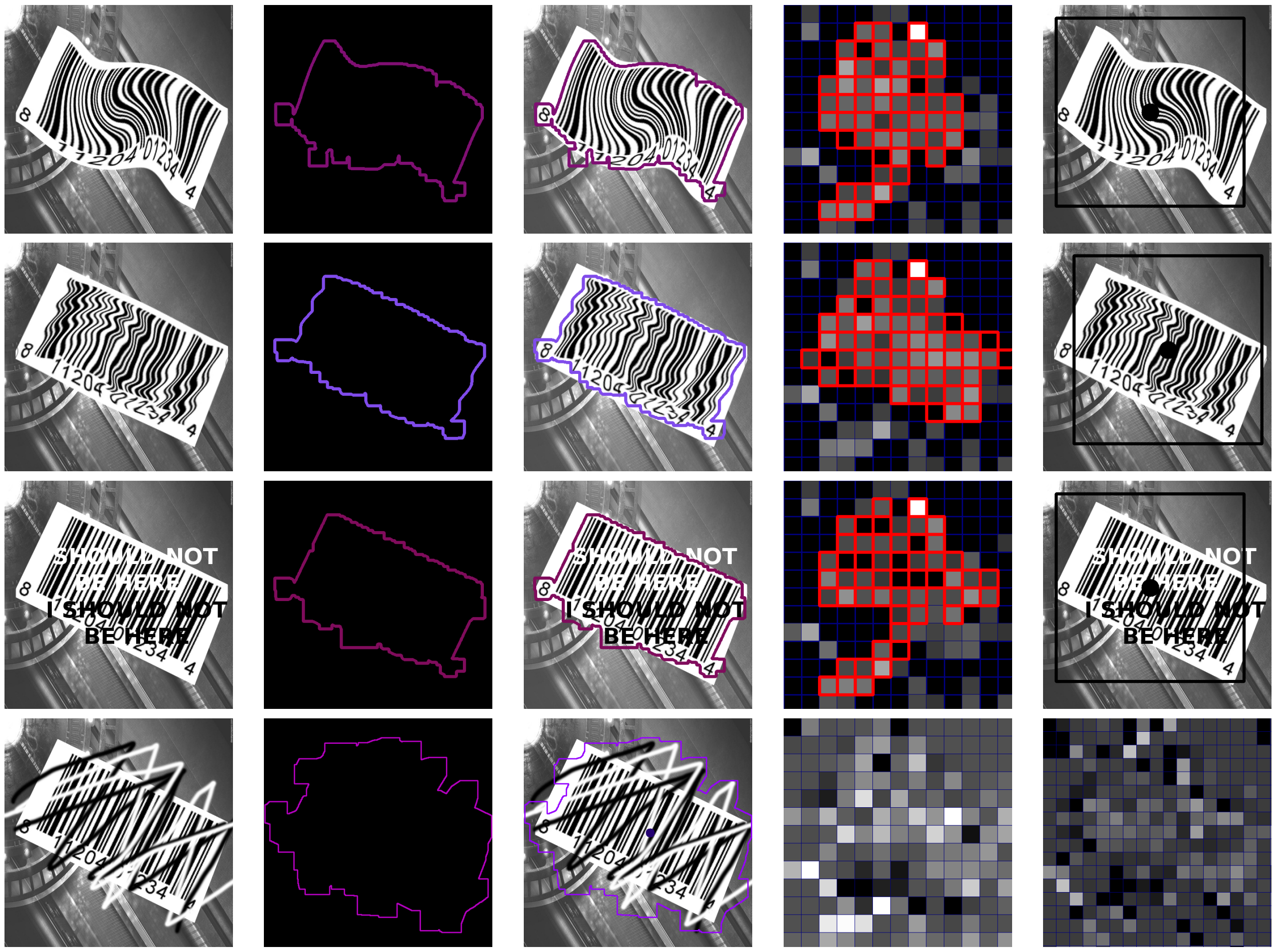


Figure 9: Test examples for MIN–MAX and Local clustering. Each row corresponds to different distortions applied to the original image. From top to bottom: twist, ripples, overlaid text, brush strokes. Images from left to right: test image, contour from the MIN–MAX feature image, test image overlaid with the MIN–MAX contour, feature matrix of Local clustering, test image with barcode center and bounding box detected by Local clustering. In the last row: Local clustering was unable to find any barcodes, even with using smaller tiles (corner image).