International Journal of Multimedia and Ubiquitous Engineering
Vol. 10, No. 1 (2015), pp. 137-248
http://dx.doi.org/10.14257/ijmue.2015.10.1.13

Decoding Deformed Barcode Images with Scanpath Walkthrough

Poonna Yospanya[1] and Yachai Limpiyakorn[2]
Department of Computer Engineering, Chulalongkorn University
Bangkok 10330, Thailand
[1]*Poonna.Y@student.chula.ac.th*, [2]*Yachai.L@chula.ac.th*

Abstract

Most camera-based barcode reading systems are scanline-based and designed to work with barcodes attached on flat or smooth surfaces. These systems are likely not able to deal with barcodes on surfaces with high level of deformation, thus making it difficult to find a scanline passing through readable regions. In this paper, an approach that addresses the decoding of deformed barcode images is presented. Our method is based on constructing a scangraph that covers over the area of the barcode, and creating a scanpath that walks through the scangraph by avoiding the regions that are less readable. Barcode digit templates are then used to decode the data sampled along the scanpath. Preliminary results show that some challenging deformed barcode images can be correctly decoded with the presented technique.

Keywords: *image processing; deformed surface; barcode, scanpath*

1. Introduction

For decades, barcodes have been widely used as a means to encode product specification in graphical formats. They are commonly used to keep track of shipments and price retail items, manage financial and logistical documents, and so on. The use of barcodes is widespread as they are compact and machine-readable.

Several types of barcodes exist, including linear or one-dimensional (1D) barcodes, and two-dimensional (2D) barcodes such as QR code. The focus of this research is on the one-dimensional barcode that encodes data of the object to which it is attached in a graphical form of bars and spaces. The special optical scanners called barcode readers are originally used for translating the structured graphical format into human-understandable data. Later, scanners and interpretive embedded software have become available on devices, such as camera phones and lightweight handheld tablet computers. These devices have become ubiquitous nowadays. They can also be considered as assistive equipment to enhance daily life for people with visual disabilities when buying consumer products.

However, most camera-based barcode reading systems are designed to work with barcodes attached on flat or smooth surfaces. This seems not practical when dealing with barcodes on surfaces with high level of deformation, containing indecipherable regions in the barcode images. On a highly deformed surface, such as on a wrinkled snack bag, reading the barcode becomes a very challenging task which likely makes all the currently available methods fail.

There are several problems that are unique to this class of barcode images. First of all, the detection of deformed barcodes is much more difficult due to the fact that elements are not uniformly aligned. In this case, unlike normal barcode images, bars in these images are warped and only partially parallel to neighboring bars. The localization methods that rely on uniformity of barcode element alignment are mostly not directly applicable.

ISSN: 1975-0080 IJMUE

As wrinkles create ridges and valleys, decoding deformed barcode images also face problems with lighting and visibility. External lighting casts reflections on ridges and shadows on valleys. Reflections appear as bright regions over the barcode, causing splits on some of its bars. Shadows make it harder to differentiate between bars and spaces due to lower local contrast, which is especially problematic for those algorithms that rely on image binarization. An example of the aforementioned problems is depicted in Figure 1.

Figure 1. Example Deformed Barcode on a Wrinkled Snack Bag (left). The Barcode is Binarized Using a Simple Threshold, Showing Splits (framed by the red square) from Light Reflection, and Merged Lines (marked by the red circle) from the camera viewpoint (right)

Superimposition may occur on ridges where one side of the ridges is not visible at the viewing angle of the camera. This may cause a total loss of information if there are bars or spaces that are completely hidden under this circumstance. However, it is also possible that parts of the covered bars and spaces are still visible and can be used for decoding. Unfortunately, bars may appear to merge with neighboring bars, further complicating the decoding process.

Furthermore, ridges and valleys cause non-uniform perspective distortion, which makes it harder to reliably determine the width of bars and spaces. Even on a single individual bar, the width can be inconsistent throughout its length. Some parts of the bar may appear wider or narrower than some other parts of the same bar.

In literature, most research work allows small deviation on the surface structure such that correct reading of barcodes can still be obtained. However, as the deviation becomes greater, correct reading of deformed barcodes becomes increasingly difficult. This paper thus presents a method of decoding barcode images on deformed surfaces that addresses some of the problems described above. The invented method is applied for the one-dimensional barcodes, especially those printed on product packages that would help the visual disabled for product purchases.

2. Related Work

A lot of research conducted in the area of camera-based barcode reading assumes that the barcode is on a nearly flat surface or a simple curved surface such as on a bottle. Earlier techniques rely on either binarization or edge detection in order to extract black and white patterns information for later decoding. Ohbuchi et al. [1] used a spiral searching method starting from the center of the image to find a black bar and establish a scanline perpendicular to the detected black bar. Bar patterns were sampled as gray levels along the direction of the scanline. These patterns were later binarized with a threshold computed during the decoding stage. Chai and Hock [2] used a binarized scanline based on the mean value to convert the cross section of the barcode as a sequence of 1’s and 0’s. The sequence was then encoded using run-length encoding which was used to compare with barcode digit patterns. Adelmann et al. [3] used multiple scanlines, each of which

was assigned with a different set of parameters and binarization thresholds. The results obtained from each scanline were then selected through a voting scheme. In other works, probabilistic approaches, such as hidden Markov models [4] and a Bayesian algorithm [5, 6], were used for estimating barcode digits based on detected edges.

Typically, the techniques based on binarization and edge detection are susceptible to noise and blur. The quality of results strongly depends on the selected threshold values. With poor selection of threshold values, it may cause narrow bars to split due to the thinning effect and bar lines close to each other to merge due to the thickening effect. Later research work has addressed these problems by taking into account the gray level of the barcode along the scanline. Gallo and Manduchi [7, 8] proposed a method based on deformable binary templates, which allow shifting and scaling of the templates to best match the scanned data. Chen et al. [9] improved the method by modeling the combining templates with some certain blur levels to better approximate the barcode data reading at a severe out-of-focus level.

3. Research Methodology

Reading the barcode in a given image generally requires both localization and decoding operations, which work synergistically in order to obtain the information encoded within. This research merely focuses on the decoding operation, relying on the input images having been stripped off of objects that are not parts of the barcode. The proposed decoding method is applied to reading the barcode images printed on deformed surfaces.

The method consists of three major stages: (1) extracting the barcode structure, (2) constructing a scangraph, and (3) decoding along a scanpath using templates.

In this work, a *scangraph* is defined as a directed graph of which the vertices are scanning fragments, connected in an orderly manner. A scanning fragment is ideally a short *scanline* that covers just a single bar or space in the barcode. A complete reading of the barcode is achieved by scanning through a set of such fragments along a path, called a *scanpath*, in the scangraph. A scanpath is derived by performing a search for a decodable path on the scangraph. More details of these concepts will be described in later sections.

In literature, the methods based on simple scanlines are successfully used for decoding slightly deformed barcode images, as there are no problems of merging lines, reflections, or splitting bars as encountered in this research. When decoding highly deformed barcode images, it is possible that the scanline that crosses over the barcode without passing through deformed areas cannot be found. The notion of scanpath addresses this problem by attempting to make a detour across the deformed surfaces and creating a path that is not strictly a single straight line.

Once a scanpath has been constructed, the scanning operation is carried out using a method based on template matching. A set of digit templates is pre-computed for comparison with the barcode patterns extracted along the scanpath. It should be noted that, although a form of binarization is applied for constructing the scanpath and segmenting the scanned data, measuring gray-level intensity is used for template matching to avoid information loss due to inappropriate threshold selection.

Figure 2 illustrates the entire decoding process presented in this paper. The elaboration of the proposed method is described in the following sections.

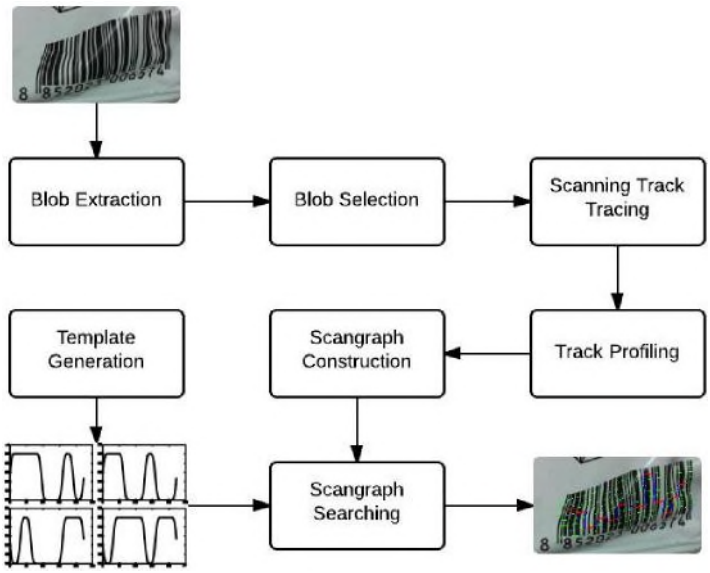

Figure 2. Proposed Method of Deformed Barcode Decoding

4. Extracting Barcode Structure

Initially, the visual structure of the barcode needs be discovered in order to create a scangraph that covers the entire length of the barcode and spans over most of its height. For a normal barcode image, the discovery process is trivial as it simply involves finding the orientation of the barcode. The structure itself can be assumed to be rectangular with vertical bars. For a highly deformed barcode, however, the structure can be of an irregular shape. In extreme cases, it is sometimes not possible to draw a straight line that passes through all the bars. Such possibility calls for a thorough shape analysis so that a scangraph that correctly follows the barcode structure could be built. In this research, a binarization technique is applied for barcode structure discovery starting with extracting the connected components called *blobs*. The extracted blobs are then examined for their suitability as structural elements.

4.1. Blob Extraction

An input image is initially converted into a grayscale image for subsequent binarization. It is assumed that the image has uneven brightness and contrast. As a consequence, a single binarization threshold cannot be used. Adaptive thresholding is thus applied with a Gaussian window function in this work. The result, however, also contains noises in the regions around the barcode as a side effect of applying a small averaging window over larger empty regions. The mask derived from applying a morphological close operation over a Canny edge image is used to help smooth out noises. Figure 3 demonstrates each operation in this step.

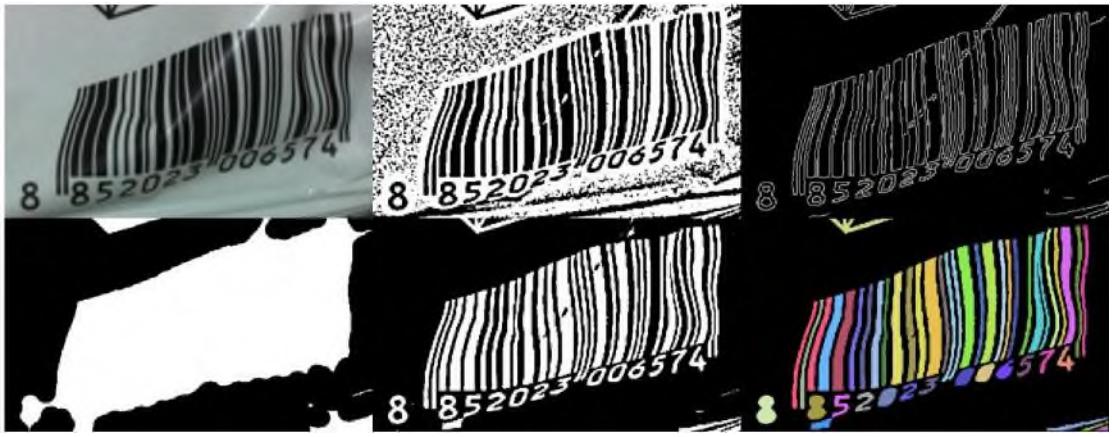

Figure 3. Output from Each Step during Blob Extraction. The Deformed Barcode Image (top left) is Binarized with an Adaptive Threshold Approach (top middle). Edge Detection using Canny Algorithm is then Performed (top right). A Mask out of the Edge Image is Created using a Morphological Close Operation (bottom left). The Resulting Mask is then Applied to the Binary Image (bottom middle). Blob Labeling is Finally Performed on the Masked Image (bottom right)

Blobs of pixels are then extracted from the binary image and consecutively labeled with a sequence number $k = \{1, ..., N\}$. Each blob b_k is a potential candidate for a bar element in the barcode. Its number is used during the decoding stage for aligning and sequencing along with other blobs. A contour C_k is also extracted for each of the blobs as a set of coordinates $\{p^k_1, p^k_2, ..., p^k_n\}$ that defines the connected line segments forming its border.

4.2. Blob Selection

Further inspection on the blobs obtained from the previous step is required to eliminate blobs which are unlikely to be one of the bar elements. Blobs that are too small are removed. Orientation and length are then used as heuristics to select the remaining blobs. The selection process is detailed as follows.

For each blob b_k, two endpoints (p^k_a, p^k_b) are selected from C_k that are at the opposite ends of the blob. It is performed by first deriving the minimum rectangle area R_k that contains all the points in C_k. Next, for each of the four corners of R_k, a point in C_k that is closest to it will be selected. These four points are then paired up to two pairs of points. Each pair corresponds to the points closest to either end point of a long side of R_k. The shortest distance along the contour for both pairs is computed. The pair with the shorter distance is selected and added into an endpoint set E. Once all the blobs have been processed to collect all endpoint pairs in E, a linear regression line L is computed from all points contained in E. For each pair (p^k_a, p^k_b) in E associated with the same blob b_k, if p^k_a and p^k_b are on the opposite side of L, b_k is selected as a viable candidate for a barcode element; otherwise, b_k is discarded. Figure 4 illustrates the method of blob selection via an example image of barcode containing split bars.

Another blob selection criterion of estimated blob length is also established. More specifically, the length l_k of blob b_k is calculated from the shortest contour path length between (p^k_a, p^k_b). Once all the lengths have been collected, the median length l_m is computed and used for specifying the threshold values. Those blobs that are either too long or too short compared to the thresholds will be discarded.

Figure 4. Blobs are Selected under an Assumption that a Valid Bar Element Should Span from Top to Bottom. The Image is Split into Top and Bottom Regions using a Regression Line. Blobs that do not Span Across both Regions are Discarded

After all the selected candidates have been collected, bar ordering is performed as follows. For each selected blob b_k, the intersection point between the line segment (p^k_a, p^k_b) and L are determined. The positions of intersection points along L are subsequently used for sorting the candidate blobs.

At the end of this step, a sorted set of blobs that are potentially bar elements is obtained. Each potential bar element is represented by a pair of endpoints and the path between them along the contour. Together they form a barcode structure that will be used to create a scangraph in the next section.

5. Constructing Scangraph

In this work, a scangraph $G_s = \{V_s, E_s\}$ is defined as a directed graph, where V_s is a set of scan fragments, and E_s is a set of edges that connect a scan fragment to one of the scan fragments that potentially follows the former in a scanning order. A scan fragment is a short scanline that crosses over a single bar or space in the barcode. A scan fragment crossing a bar is connected to a scan fragment crossing the space next to the bar, and vice versa.

G_s covers almost the entire region of the barcode. To construct G_s, multiple scanning tracks along the height of the barcode are firstly created. Each track will then be scanned in order to collect a track profile, which consists of the blob number and the relative location on the track each blob passes through. These track profiles are used as the foundation for constructing the scangraph.

5.1. Scanning Track Tracing

The contour path between each pair (p^k_a, p^k_b) of a selected blob b_k from previous step is divided into $\lfloor l_p/l_s \rfloor = M_k$ segments of length l_s, where l_p is the length of the contour path and l_s is a pre-determined segment length, producing a set of M_k+1 anchor points $\{q^k_1, q^k_2, \ldots, qkMk+1\}$ for each blob b_k. Ideally, anchor points of the same rank i from each of the blobs are threaded together in the scan ordering determined in the previous step to form a scanning track T_i, which is a sequence of line segments passing through anchor points $\{q^1_i, q^2_i, \ldots, q^N_i\}$ and crossing over all the blobs. In fact, some anchor points of the same rank on different blobs can stray far away from the ideal alignment. Substituting a strayed anchor point with another one of a different rank from the same blob is allowed if it is more in line with other anchor points, subject to a penalty distance function. Figure 5 demonstrates a selection of such scanning tracks as a result of this step.

Figure 5. Demonstration of Scanning Track Tracing. Anchor Points are Created on the Barcode Structure (middle), and Used to Generate Scanning Tracks (right)

5.2. Track Profiling

Each scanning track T_i is scanned, and at each entry and exit event of a blob, the blob number and the corresponding track position are recorded. The recorded profile is in the form of a sequence of tuples $\{(b_k, p^{Ti}_j) \mid j \in 1..N\}$, where N is the number of events in the profile. A line segment (p^{Ti}_j, p_{Tij+1}) between two consecutive events is called a scanning fragment. This line segment crosses over an element of the barcode. The crossed-over element can be either a bar or a space. Each bar or space contains a number of scanning fragments equivalent to the number of scanning tracks. These fragments represent vertices in the scangraph being constructed.

5.3. Scangraph Construction

Intuitively, the scangraph connects each fragment of a bar to all other fragments of the space following the bar, and each fragment of a space to all other fragments of the bar following the space. Ideally, the process of constructing such graph should be trivial. There are, however, a number of practical issues that make the process more difficult. Due to high level of deformation, multiple bars can appear merged together at some points into a single bar. In this case, some scanning tracks will register only one bar whereas some others will register multiple bars with the same blob number. In addition, reflections can obscure parts of some bars in the barcode, making them visually split into multiple pieces. It is observed that some scanning tracks may not register a bar when it should, or may register the bar twice if the split occurs at a sharp angle such that the track passes through both parts of the split. There are also problems with noises which create spurious blobs, but this can be mitigated as long as there are some tracks that do not pass through them.

Figure 6 shows the algorithm for constructing a scangraph that proceeds on a track-by-track basis. That is, for each track, the algorithm walks through each fragment on the track sequentially, trying to find suitable successor fragments on all tracks for that fragment. In this way, a fragment from a track t can branch to a next fragment on any other track s. Branching is computed differently for fragments that are bars and fragments that are spaces. A bar fragment q is considered a successor of a space fragment p if at least one of the following conditions is true:

(1) It is a direct successor of p on the same track,

(2) It is on the same blob and at the same depth on that blob as the direct successor of p,

(3) It is the first successor of q' that is on a different blob from that of p', where q' is a successor of p' on track s, and p' is the direct predecessor of p on track t.

Cases (2) and (3) deal with blobs that contain merging bars. Suppose bar a and bar b merge to form a blob c, fragments from a and b will belong to the same blob. Case (2)

attempts to connect a space fragment between *a* and *b* to a bar fragment in *b*. Case (3) avoids using fragments from the same blobs in the same path. These two cases are complementary to cover most possible scenarios.

For branching from a bar fragment to space fragments, it simply takes the branching from its direct same-track predecessor and moves forward one fragment along their corresponding tracks. For example, when branching from a bar fragment *p* having a space fragment *q* as its direct same-track predecessor, all the successors of *q* are considered to find their direct same-track successors. These successors of successors become the branching successors of *p*.

```
function BranchesAfterSpace(t, p):
    branches := {}
    nextCode := BlobNumber(t, p+1)
    for each track s:
        if s = t:
            Add(branches, (s, p+1))
        else if nextCode exists in trackProfiles[s]:
            n := number of occurrences of nextCode before p
            q := n^th fragment having same blob number as nextCode
            Add(branches, (s, q))
        else:
            q' := fragment on branch to s at G[t, p-1]
            previousCode := BlobNumber(s, q')
            q := first fragment after q' whose blob number ≠ previousCode
            Add(branches, (s, q))
    return branches

function BranchesAfterBar(t, p):
    branches := {}
    previousBranches := all branches at G[t, p-1]
    for each branch b in previousBranches:
        s := Track(b)
        q := Fragment(b)
        Add(branches, (s, q))
    return branches

function BuildScangraph():
    G := {}
    for each track t:
        for each fragment p on t:
            if trackProfiles[t, p] is a space:
                G[t, p] := BranchesAfterSpace(t, p)
            else:
                G[t, p] := BranchesAfterBar(t, p)
    return G
```

Figure 6. Algorithm for Scangraph Construction

6. Decoding along a Scanpath using Templates

To decode the barcode, the scangraph is searched for a path with high template matching score, then each digit is individually decoded within the scanpath using pre-computed templates. Prior to describing the algorithm for finding a scanpath, the process of template matching for each digit will be discussed first in the following sub-section.

6.1. Template Matching

Unlike previous steps that work on the binary image of the barcode, this step uses the gray-scale image for digit decoding. A digit along the scanpath P is a subpath P_d containing exactly four consecutive fragments. To decode a digit, $_{1V,,}$ samples are taken from the barcode image for gray-level intensity at equally spaced points along P_d at subpixel level, where $_{1V,,}$ is the number of samples in each digit template. In this work, Pearson correlation coefficient is used as the similarity metric to compare digit samples against those of each digit template.

The templates used in this step are pre-generated directly from the barcode digit encoding table, with the zero value representing bars and the one value representing spaces. The resulting templates are then smeared slightly using a Gaussian filter to correspond with the effect of edge blurring in real photographs. Figure 7 depicts some examples of the final templates and a sampled digit.

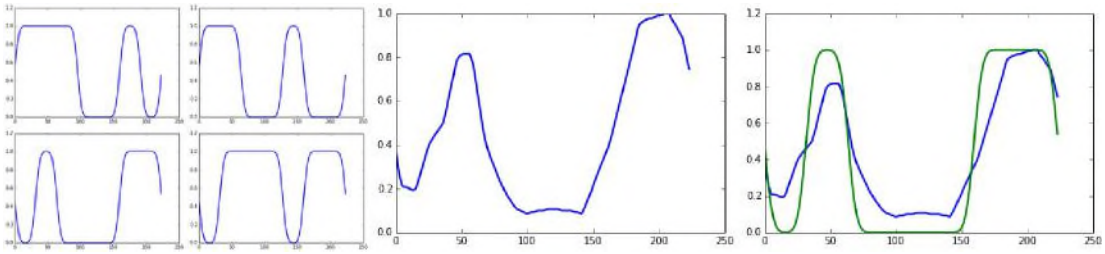

Figure 7. Templates are Generated from Barcode Encoding and Smeared using a Gaussian Filter to More Closely Match the Blur Effect from the Camera. The four Templates Shown above are of the Odd-parity digit 0, even-parity digit 2, no-parity digit 4, and no-parity digit-7, respectively (left). To Perform Matching along a Scanpath, a Subpath is Sampled (center), and Evaluated using Pearson Correlation Coefficients Against all Applicable Templates. A Match with the No-Parity Digit-4 Template is illustrated (right)

6.2. Scangraph Searching

Referring to EAN-13 barcodes, bars and spaces are grouped into digits and guard patterns. Each barcode digit contains two bars and two spaces, while guard patterns at the left and the right contain two bars and one space each. Guard pattern at the center contains two bars and three spaces. There are a total of 13 digits, split into 7 left digits and 6 right digits. The first digit of the left group is computed from parity values of the following 6 digits. Digits of the right groups have no parity, but the last digit is used as the checksum for the whole 13 digits. The specification of EAN-13 barcodes as described above is used for pruning and verifying the search on the scangraph.

In actual decoding, template matching is performed alongside the scangraph search and it is used to determine the next set of fragments being branched to. The actual branching starts after the guard bars. The depth-first search will start at a first fragment of the first bar by skipping the first three fragments that are parts of the left barcode guard. For each point in the scangraph, there are several successive fragments to branch to. For each possible branch, the four consecutive same-track fragments on that branch are grouped as a potential digit, and the correlation coefficients between them and each possible digit template are computed. The operations are repeatedly performed for every branch to collect all the correlation coefficient pairs. The one with the highest correlation coefficient

is then selected as the digit candidate and the chosen branch. After the four fragments on the chosen branch, the process is repeated to find the branch for the next digit.

Once having repeated the process for six digits, the parity values of these six digits is used to look up the first digit on the parity table. If there is a match, the match will be selected as the first digit. At this point, the process is advancing through the center guard of the barcode, and it will continue the same branching strategy as performed on the left half. If there is no match for the parity values, however, the search is backtracked and the next best branch is chosen.

When the scanpath reaches the final digit, all the digits are tested against the checksum. If passing the test, the process reports the decoding result and terminates. Otherwise, the search is backtracked and attempts another path and/or another decoded digit with the next best correlation coefficient. If all possible scanpaths are exhausted and it still cannot find a valid decoding, the search is declared a failure. It is recommended that branching should be limited to only a few top-scoring paths, as the finding from our experiments reports that low scoring paths rarely produce a better result. An example of a valid scanpath is shown in Figure 8.

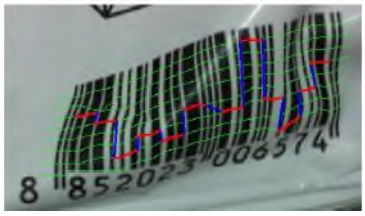

Figure 8. Example Scanpath that Produces correct Decoding Result. Red Line Segments Show the Detected Fragments, Connected by Blue Line Segments

7. Preliminary Results

The proposed method has been implemented with Python 2.7.7, using OpenCV 2.4.9 library for image processing tasks. A set of deformed barcode images has been created as the tests in the preliminary experiments. Figures 9, 10, and 11 depict the test results in various scenarios using our own images. Figure 12 reports the result of testing using an image from the barcode images database [10].

Starting from testing with five different deformed barcodes, some of which are highly deformed, i.e. containing merged bars and light reflections. All correct decoding results are obtained as shown in Figure 9. Notice that the chosen scanpaths automatically avoid irregularities such as reflections and merging. The sub-paths containing these irregularities generally produce low correlation coefficients. They are thus excluded from being parts of the selected scanpath.

In some cases, the presented technique may produce no result as shown in Figure 10. The example exhibits a high degree of merging and binarization errors that cause the scanning algorithm to miss some elements. Figure 11 demonstrates the case of incorrect decoding result. Notice that the algorithm erroneously skips a bar when decoding the last four digits. This is caused by an error in the binarization step.

There are still some issues that need to be addressed. Currently, a track profile is required as a guide when advancing through each fragment. Note that the track profile is derived from blob labeling, which is subject to binarization error. This may cause the algorithm to skip some bars or spaces, producing incorrect results as shown in Figure 10 and Figure 11. Moreover, the current method does not work well with blurry and low-contrast images. As illustrated in Figure 12, improper threshold values cause Canny edge

detector to miss most of the edges, which in turn, produces a mask image that does not cover the barcode well. This results in the binary image with a lot of elements missing. The following steps of track tracing and profiling then produce invalid output as a consequence.

8. Conclusion

The scanpath technique is introduced in this work as a means of barcode decoding that can work under high deformation condition, which, to the best of our knowledge, has not previously been specifically explored. The presented approach introduces the use of the non-linear scanline, so called scanpath, to trace through the deformed barcode image attempting to avoid parts that are damaged by reflections and hidden elements. A binarization technique is adopted to provide cues for the detection of reflections and hidden elements, and to create anchor points for the scanpath to tread through. However, binarization is not used during the actual decoding. A method based on template matching is applied for decoding instead. The work is ongoing and preliminary results are promising. There are still a number of challenging issues. Future research direction would include the refinement on the construction of scangraph, the enhancement of a more resilient method for template matching that relies less on binarization, and the improvement on the part of localization to manage noises, blurs, poor lighting conditions, and low contrast.

Figure 9. Test Results on Five Different Deformed Barcodes that are all Successfully Decoded

Figure 10. High Degree of Bars Merging in the Middle Part of the Barcode Causes the Algorithm to Miss Some Elements

Figure 11. Binarization Error Caused by Shadow Regions Makes the Algorithm Skip a Bar Element and Produce an Incorrect Decoding Result

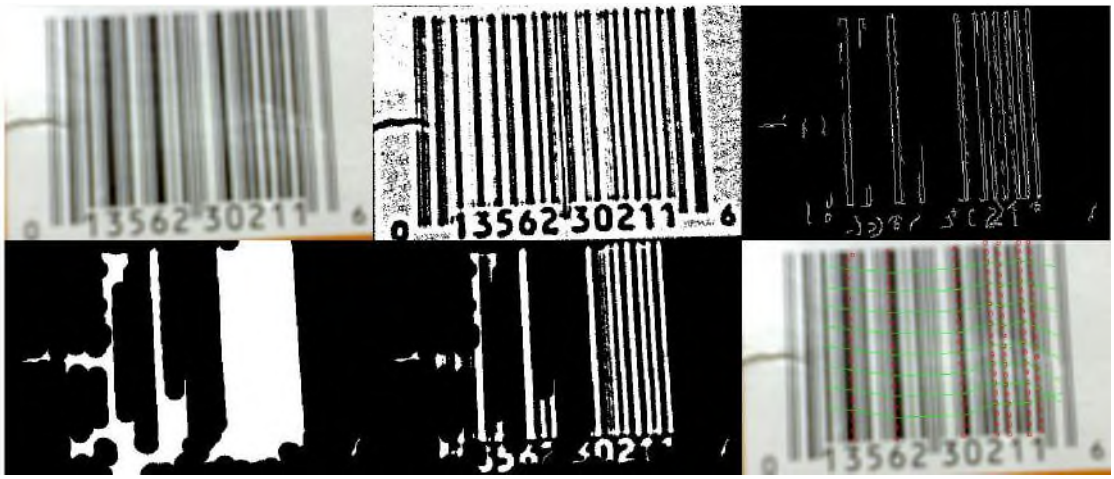

Figure 12. Current Method does not Handle Images with Blurs or Low Contrast (upper left). In this example, a Failure in Edge Detection (upper right) Produces an Incomplete Mask Image (lower left), Resulting in Missing Blobs (lower middle) and Incomplete Track Tracing (lower right)

9. References

[1] E. Ohbuchi, H. Hanaizumi and L. A. Hock, "Barcode readers using the camera devices in mobile phones", Int. Conf. on Cyberworlds, **(2004)**.
[2] D. Chai and F. Hock, "Locating and decoding EAN-13 barcodes from images captured by digital cameras", Int. Conf. on Information, Communication and Signal Processing, **(2005)**.
[3] R. Adelmann, M. Langheinrich and C. Flörkemeier. "A toolkit for bar code recognition and resolving on camera phones – jump starting the internet of things", Workshop Mobile and Embedded Interactive Systems (MEIS) at Informatik, **(2006)**.
[4] S. Krešić-Jurić, D. Madej and F. Santosa, "Applications of hidden Markov models in bar code decoding", In Pattern Recognition Letters, vol. 27, no. 14, **(2006)**, pp. 1665-1672.
[5] E. Tekin and J. Coughlan, "A Bayesian algorithm for reading 1D barcodes", Canadian Conf. on Computer and Robot Vision, **(2009)**.
[6] E. Tekin and J. Coughlan, "An algorithm enabling blind users to find and read barcodes", IEEE Workshop on Applications of Computer Vision (WACV), **(2009)**.
[7] O. Gallo and R. Manduchi, "Reading challenging barcodes with cameras", IEEE Workshop on Applications of Computer Vision (WACV), **(2009)**.
[8] O. Gallo and R. Manduchi, "Reading 1-D barcodes with mobile phones using deformable templates", In IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 33, no. 9, **(2010)**, pp 1834-1843.
[9] L. Chen, H. Man and H. Jia, "On scanning linear barcodes from out-of-focus blurred images: a spatial domain dynamic template matching approach", in IEEE Trans. on Image Processing, vol. 23, no. 6, **(2014)**, pp. 2637-2650.
[10] Barcode images database, [Online]. Available: http://www.ski.org/Rehab/Coughlan_lab/Barcode/, **(2009)**.

Authors

Poonna Yospanya. He received his bachelor degree in Computer Engineering from Kasetsart University in 1999. After graduation, he had worked as a systems engineer at KSC Commercial Internet for two years. He has been working as a lecturer at the Department of Computer Engineering, Kasetsart University, Sriracha Campus, since 2001. He is currently pursuing a Master degree in Computer Science at Chulalongkorn University, Bangkok, Thailand.

