THE SMITH – KETTLEWELL
EYE RESEARCH INSTITUTE
2318 FILLMORE STREET • SAN FRANCISCO • CA • 94115

THE SMITH-KETTLEWELL EYE RESEARCH INSITUTE
REHABILITATION ENGINEERING RESEARCH CENTER

TECH REPORT 2012-RERC.01

BLaDE: Barcode Localization and Decoding Engine

Ender Tekin

James M. Coughlan

December 12, 2012

1 Introduction

This technical report presents details of the computer vision algorithms and user interface (UI) features implemented in BLaDE (Barcode Localization and Decoding Engine), which is a system designed to allow blind and visually impaired persons find and read product barcodes. BLaDE has been implemented both as a Linux desktop application (which requires the use of a webcam) and as an Android smartphone app. Experiments with the BLaDE smartphone app tested by blind and visually impaired users are described in [1], and additional context about BLaDE (including how it compares to other barcode readers) is included in [1, 2].

The primary innovation of BLaDE, relative to commercially available smartphone apps for reading barcodes, is that it provides real-time audio feedback to help visually impaired users locate a barcode, which is a prerequisite to being able to read it.

Currently the only barcode symbology supported by BLaDE is UPC-A (see Sec. 3.1), which is the most common symbology used to label commercial products in North America. However, it would be straightforward to extend BLaDE to detect and read other symbologies.

BLaDE is an open source project released under the BSD License: code is available at `http://sourceforge.net/p/sk-blade`.

The BLaDE library code only returns the decoded UPC-A number. Once the barcode is decoded, the product can be looked up using an online service such as Google Product Search, which is implemented in the client software released for Linux and Android platforms.

2 Barcode Detection & Localization

Other barcode-based product identification apps make the assumption that the barcode is carefully centered in the image, or at least is well oriented and visible in the image. This assumption is not necessarily true for visually impaired users, requiring an additional step that involves detection of the barcode at various orientations, as well as sizes. This information can then be communicated to the user in the form of audio feedback to help direct the user to point the camera at the barcode such that it can be resolved with sufficient quality to enable decoding.

To ensure usability of this algorithm by a visually impaired user, it is essential that this process can

1. run at real-time and with minimal delay, to allow the user to efficiently scan possibly large objects;
2. communicate with minimal delay, such that the information about the possible location of the barcode is still applicable while the user is moving the camera during the scan.

We have developed an approach that utilizes the grayscale video image to extract edge orientations and decide on the location and extent of a barcode *scanline*, defined as a cross-sectional segment that covers the whole barcode, as in Fig. 1. The steps of this algorithm are outlined next:

Figure 1: A barcode stripe detected by BLaDE.

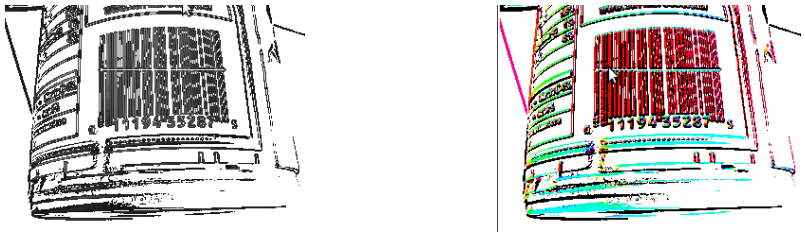

Figure 2: Barcode image gradients – left: magnitudes, $m(x)$, right: orientations, $o(x)$.

2.1 Pre-processing

The main feature of a barcode that we utilize during the barcode detection/localization stage is that *a barcode contains many parallel edges and no other edges*. As a first step, the edges in the image, defined as the points where the intensity gradient magnitude is greater than a fixed threshold, need to be determined.

To accomplish this, the video frame is first converted to grayscale. Next, the image gradients are calculated using a Scharr operator which has good rotational symmetry (useful for extracting edges at various orientations), while still being separable (requiring less number of operations than computation speed) [3]. The Scharr operators for vertical and horizontal gradients are:

$$S_x = \begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} 3 \\ 10 \\ 3 \end{bmatrix} \tag{1a}$$

$$S_y = \begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} = \begin{bmatrix} 3 & 10 & 3 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \tag{1b}$$

Then, x and y gradients of image I are given by

$$\nabla_x I = S_x * I \tag{2a}$$

$$\nabla_y I = S_y * I \tag{2b}$$

where $*$ denotes convolution.

The x and y gradients are then converted to polar coordinates. To reduce computational load and eliminate noise, the gradient magnitude is thresholded such that for pixel x,

$$m(x) = \begin{cases} |\nabla I(x)|, & \text{if } |\nabla I(x)| > \tau_{\text{mag}} \\ 0, & \text{otherwise} \end{cases}. \tag{3}$$

where τ_{mag} is a threshold on the minimum gradient magnitude.

To be able to detect barcodes at a fine enough angular resolution, $N_o = 18$ different orientations covering $[0, \pi)$ radians are analyzed. To achieve this, the gradient angles are quantized at this stage to to one of $2N_o = 36$ bins, each covering 10 degrees, giving

$$o(x) = \left\lfloor \frac{\angle \nabla I(x)}{\pi / N_o} + \frac{1}{2} \right\rfloor \tag{4}$$

such that $o(x) \in [0, 2N_o)$.

2.2 Determining Orientation

To detect possible barcode areas, a voting scheme is used. The image is divided into non-overlapping 20pixel$\times$ 20pixel patches. For each patch $P^{(k)} \in \mathcal{P}$, where $\mathcal{P}$ is the set of all patches, the following steps are taken:

1. The histogram of collapsed orientations is calculated, where each pixel casts a weighted vote for barcode orientation $\omega \in [0, N_o)$ corresponding to $[0, \pi)$. The weight of a pixel's vote is the thresholded gradient magnitude at that pixel, $m(x)$, to emphasize the contribution of strong edges and reduce noise. In patch k, the bin for orientation i thus has

$$\omega_i^{(k)} = \sum_{x \in P^{(k)} | o(x) = i} m(x) + \sum_{x \in P^{(k)} | o(x) = i + N_o} m(x), \quad \forall i = 0, \ldots, N_o - 1 \tag{5}$$

   votes, coming from positive and negative edges in the ith quantized orientation.

2. The *peakedness* of the orientation histogram is measured using Shannon entropy. The entropy of the resulting histogram is given by

$$H(\boldsymbol{\omega}^{(k)}) = -\sum_{i=0}^{N_o - 1} \frac{\omega_i^{(k)}}{\omega_\Sigma^{(k)}} \log \frac{\omega_i^{(k)}}{\omega_\Sigma^{(k)}}, \quad \text{where } \omega_\Sigma^{(k)} = \sum_{i=0}^{N_o - 1} \omega_i^{(k)}. \tag{6}$$

   A patch that is part of a barcode should provide a dominant orientation, determined by a patch gradient orientation histogram having low entropy. If the entropy $H(\boldsymbol{\omega}^{(k)}) > \tau_{\text{entropy}}$, where τ_{entropy} is a fixed threshold, then this patch is unlikely to be part of a barcode, and is discarded as a barcode candidate. Otherwise, the patch is saved for further processing.

Let the set of saved patches to be $\mathcal{P}' = \left\{ P^{(k)} \in \mathcal{P} \mid H(\boldsymbol{\omega}^{(k)}) < \tau_{\text{entropy}} \right\}$. If there is a barcode in the image at orientation $o \in [0, N_o)$, it should manifest as gradients at o and $o + N_o$. To ensure that the peaks at opposing polarities are easily resolved, we form the global orientation histogram $\boldsymbol{\Omega}$ such that

$$\Omega(i) = \min \left\{ \sum_{P^{(k)} \in \mathcal{P}'} \sum_{x \in P^{(k)} | o(x) = i} m(x), \sum_{P^{(k)} \in \mathcal{P}'} \sum_{x \in P^{(k)} | o(x) = i + N_o} m(x) \right\}, \quad \forall i = 0, \ldots, N_o - 1, \tag{7}$$

thus suppressing single strong edges that may show up in the histogram without an edge pattern of alternating orientations.

Next, Kernel Density Estimation (KDE, [4]) is applied to $\boldsymbol{\Omega}$, followed by gradient ascent to identify the modes of this histogram, using simple line search to identify a good step size. This algorithm is modified to take into account the rotational symmetry of the orientations, such that when calculating the gradient, the distance between two points i, j is calculated as $\min\{|i - j|, |j - i + N_o|\}$. The modes of $\boldsymbol{\Omega}$, denoted as $\mathcal{O}$, are considered as the set possible orientations of barcodes in the current frame.

2.3 Localization

The next step after determining the orientation is to determine the location(s) of possible barcodes. Using the patches $P^{(k)} \in \mathcal{P}'$ as points to be clustered, mean-shift clustering (see Reference [5]) using a Gaussian kernel is used to find possible barcode centers. For each orientation mode $O \in \mathcal{O}$, a separate weighted mean-shift clustering, is performed using the centers of the patches with this dominant orientation. In this step, the dominant orientation of each saved patch is determined by looking at the orientation with the most votes in

that patch, denoted $O^{(k)}$ for patch k; all patches $P^{(k)}$ such that $O^{(k)} = \lceil O \rceil$ or $O_k = \lfloor O \rfloor$ vote with a weight equaling the number of pixels with non-zero gradients in that patch. It was empirically determined that this gives a better result than using the total gradient which was used for within-patch voting, as illumination differences between patches seemed to skew the results towards those with stronger edges.

For orientation O, the determined cluster centers are taken as likely locations of barcodes, provided that the number of total votes (weight) at that point is above a fixed threshold. The detected set of barcode locations at orientation O is denoted by $\mathcal{X}_O$. Experimentally, it was seen that by this point, there is usually only one candidate barcode candidate, if any.

As a final verification step, a line is swept at a direction perpendicular to O for each barcode location candidate $X_O \in \mathcal{X}_O$. Starting at the detected barcode location, and going in opposite directions, the algorithm looks for alternating edges oriented at $[O-2, O+2]$ and $[O+N_o-2, O+N_o+2]$ within a number of pixels, to allow for a slop factor of ± 20 degrees in the barcode orientation. A score of the number of such edges is kept, and if no alternating edge is found within a threshold distance, or edges at other orientations are determined, this score is reduced. When the score reaches zero, or the edge of the frame is encountered, the maximum score is compared to a threshold. If the maximum score of this scanned segment exceeds the threshold, this image segment is saved as a verified barcode segment, $\mathbf{b}$.

This sweep is performed at each orientation mode $O \in \mathcal{O}$, and the resulting verified barcode segments are denoted $\mathcal{B} = \{\mathbf{b}^{(k)}\}_{k=1}^{N_B}$, where N_B is the total number of detected barcodes.

2.4 Audio Feedback

To avoid conflicting user feedback, feedback regarding only a single barcode is provided to the user. It is deemed that the most visible or likely barcode segment $\mathbf{b}^{(k)} \in \mathcal{B}$ should be the one with the highest number of edges. Then, audio feedback is provided regarding the location of this detected segment, which we denote $\mathbf{b}^*$.

The feedback encompasses two orthogonal directions:

1. The *size* of the barcode, indicating whether to move closer or farther to be able to resolve the barcode. Ideally, it was deemed that the barcode should lie within 60% and 80% of the image. To help the user orient the camera to satisfy this requirement, a feedback score is calculated as

$$F_{\text{size}} = \begin{cases} 1.0, & \text{if } w_{\min} \le w \le w_{\max} \\ w_{\max}/w, & \text{if } w > w_{\max} \\ w/w_{\min}, & \text{if } w < w_{\min} \end{cases}, \tag{8}$$

   where w is the width of the segment, and $w_{\min}, w_{\max}$ are calculated as 60% and 80% of the frame's minimum dimension, respectively.

2. The *alignment* of the barcode, indicating whether the barcode is deemed to be fully visible in the image, by considering how far the detected barcode is from the edges of the frame. Letting d_{min} be the minimum allowed distance from an edge (usually set at 10% of the minimum dimension), and the feedback score is calculated as

$$F_{\text{alignment}} = \alpha_{\text{left}} \alpha_{\text{right}} \alpha_{\text{top}} \alpha_{\text{bottom}}, \tag{9}$$

   where

$$\alpha_{\text{side}} = \min\left\{ \frac{1}{2}\left(\frac{d_{\text{side}}}{d_{\min}} + 1 \right), 1 \right\}, \text{ where side} \in \{\text{left}, \text{right}, \text{up}, \text{down}\}. \tag{10}$$

and d_{side} is the minimum distance, in number of pixels, of the detected barcode stripe to a given side of the image.

These feedback values are then used to modulate the amplitude and duty cycle of an audio signal. For speed, these values are quantized to $0, 1, \ldots, 5$ by multiplying by 5 and rounding, and use pre-recorded audio snippets. The audio-snippets are $T = 50ms$ long to reduce delay (the circular audio buffer holds 4 frames, such that the delay is only 200ms), and the corresponding audio signal is

$$a(F_{\text{alignment}}, F_{\text{size}}, t) = \begin{cases} F_{\text{size}} \cos(2\pi f t), & \text{if } t < \frac{F_{\text{alignment}}}{5} T \\ 0, & \text{otherwise} \end{cases}, \quad 0 \leq t < T, \tag{11}$$

and f is chosen to be a suitable frequency, in our case 800Hz.

3 Decoding the Barcode

3.1 UPC-A Symbology

UPC-A codes consist of 11 numerical digits ($s_1, s_2, \ldots, s_{11}$) + 1 check (parity) digit (s_{12}), using a many-width symbology. Each symbol consists of two bars and two spaces of widths that are an integer multiple of Δ, the fundamental width; and each symbol is exactly 7Δ wide. Each UPC symbol is represented by a 7-bit pattern as shown in Table 1. The 12 symbols are broken into two groups of 6 symbols. For the symbols before the middle bars, 1 indicates a stripe, and 0 indicates a gap. After the middle bars, this is switched, so that 1 corresponds to a gap, and 0 corresponds to a stripe. This arrangement and an example barcode is shown in Fig. 3, and the symbols coding the digits are presented in Table 1.

Figure 3: UPC-A 012345-678905. The digits are organized as $Ss_1s_2s_3s_4s_5s_6Ms_7s_8s_9s_{10}s_{11}s_{12}E$ where S and E are the start and end characters, represented by the 3-bit pattern 101, and M is the middle 5-bit guard bar 01010.

The parity digit is calculated such that 3 times the sum of the odd symbols plus the sum of the the even symbols is exactly divisible by 10, i.e.,

$$\left(3 \times \sum_{i \text{ odd}} s_i + \sum_{i \text{ even}} s_i \right) \mod 10 = 0. \tag{12}$$

3.2 Determining Symbol Boundaries

Having determined a strip of the barcode, which is called a scanline, a simple convolution based approach can be used to decode the barcode. Given the detected stripe, an initial estimate of the fundamental width of the barcode, Δ_{est}, is given by:

$$\Delta_{\text{est}} = \frac{w}{95}. \tag{13}$$

Digit	7-bit pattern	Alternate form
0	0001101	3-2-1-1
1	0011001	2-2-2-1
2	0010011	2-1-2-2
3	0111101	1-4-1-1
4	0100011	1-1-3-2
5	0110001	1-2-3-1
6	0101111	1-1-1-4
7	0111011	1-3-1-2
8	0110111	1-2-1-3
9	0001011	3-1-1-2
S,E	101	1-1-1
M	01010	1-1-1-1-1

Table 1: UPC bar code patterns. The alternate representation indicates the width of the bars between switches between gaps/bars.

For computational reasons, the detected scanline $\mathbf{b}^*$ is up-sampled such that its fundamental width is exactly $\Delta = 10$ pixels using linear interpolation, and the mean of the scanline intensity is subtracted. We denote the resulting vector as $\boldsymbol{\beta}$. As the expected location of each digit (i.e., digit 1 should start at 3Δ pixels from the first edge, digit 2 should start at 10Δ pixels from the edge etc) can be estimated, by calculating the inner product of an ideal digit pattern (consisting of 1's and -1's and fundamental width Δ) at the expected locations of a symbol, the most likely digit for a particular symbol can be determined.

However, due to perspective transformations, as well as non-flat surfaces that may be hosting the barcode, these estimates can be inaccurate. To make this estimate more robust, we use a Bayesian model to estimate the locations of the fixed edges of the barcode, which are the edges of the guard bands, as well as the edge between different symbols, and are independent of the actual coded symbols. The 24 fixed edges in a UPC-A barcode are illustrated in Figure 4. After determining these edges, and hence the symbol boundaries, the digits coded by each symbol can be decoded more accurately by this convolutional approach.

Figure 4: Fixed edges of UPC-A barcode shown as dashed red lines. Labels on bottom denote the guard regions and the 12 digit regions between the fixed edges.

To determine the fixed edges, an elongated edge operator of the form

$$\mathbf{e} = [1, 1, 1, 1, 1, -1, -1, -1, -1, -1]$$

is used to extract all possible barcode edges in the scanline. This operator of width $\Delta = 10$ helps eliminate stray thin edges that may otherwise show up in a simple gradient operation due to shadows, image artifacts, etc., as shown in Figure 5. We define the output of this edge operator applied to the barcode stripe at pixel x as $F(x) = \boldsymbol{\beta} * \mathbf{e}$, where $*$ denotes the convolution operator.

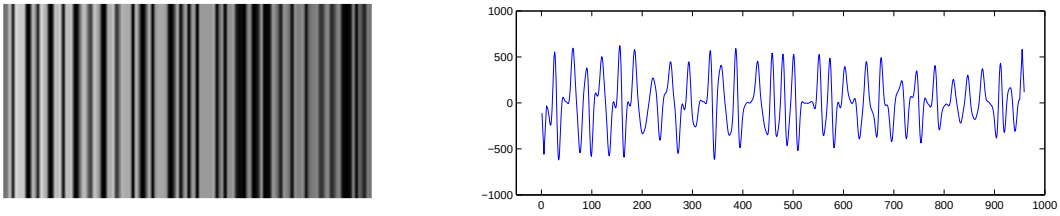

Figure 5: Left: Extracted barcode stripe $\boldsymbol{\beta}$ rendered as a 2D image. Right: Output of the edge filter, $F(x)$. Note the spurious edges that show up as small peaks in the filter output near 0. These are easily removed by thresholding.

We then formulate a Bayesian model of the edge locations, by assuming that the detected edges are noisy readings of the actual edge locations, where the additive noise is modeled as a Gaussian, i.e.,

$$E_i - E_{i-1} \sim \mathcal{N}\left(\delta_i, \sigma^2\right), \ \forall i = 1, 2, \ldots, 23. \tag{14}$$

where E_i is the ith fixed edge, and δ_i is the actual expected distance between the ith and $(i-1)$st fixed edge. σ is a parameter that was chosen experimentally to give the most number of accurate readings in the database (also released with a Creative Commons attribution license). For within-guardband edges, $\delta_i = \Delta$, and for symbol boundaries, $\delta_i = 7\Delta$. Figure 6 shows a graphical representation of the fixed-edge model within a UPC-A symbology:

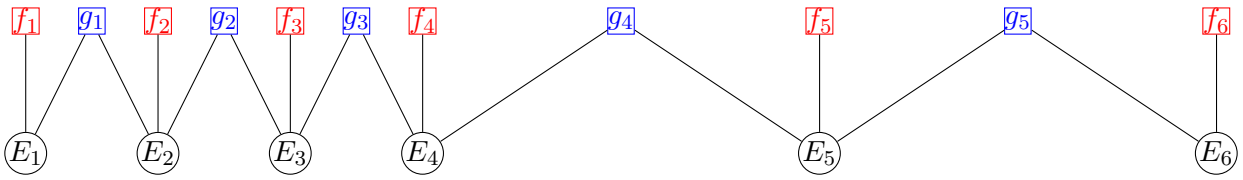

Figure 6: Graphical model representing the first 6 fixed edges in a UPC-A barcode. The first four edges $(E_1, \ldots, E_4)$ correspond to the left guard band, and the next two (E_5, E_6) correspond to the boundaries between symbols 1&2 and 2&3.

Determination of the fixed edges can then be regarded as finding the most likely set of edges among all detected edges based on the edge distribution given above. This can be written as an energy minimization problem:

$$\arg\max_{E_i} \sum_{i=1}^{24} f(E_i) + \sum_{i=1}^{23} g_i(E_i - E_{i-1}) \tag{15}$$

where

$$f(x) = c_1 \max(F_{\max} - |F(x)|, 0), \tag{16}$$

$$g_i(x) = \frac{1}{\sigma^2}(x - \delta_i)^2 \tag{17}$$

Here, $f(x)$ enforces edge strength based on the edge filter output, and $g_i(x)$ enforces pairwise distances between the edges. $F_{\max}$ is a parameter chosen experimentally as 200 to suppress weak edge detections that are generally false detections, as well as limit the influence of very strong edges. We initially added a third term that penalized edges that are very far from their expected locations based on the estimate of the

barcode's fundamental width and first edge. However, experimentally, we found that enforcing such absolute edge locations resulted in erroneous fixed edge determinations, especially for barcodes that are printed on non-flat surfaces. As such, the penalty due to this constraint was removed by setting its coefficient to zero in the released code.

By enforcing the constraint that all fixed edge must have a certain number of positive and negative edges (depending on their particular index) before and after them in the barcode strip, as well as the trivial constraint that edge E_{i+1} must lie to the right of edge E_i, the number of candidate edges for each fixed edge location can be trimmed to a small number. The problem then reduces to finding the most likely combination of edges that minimize the energy function given in (17). Since the edge model only has pairwise interactions, the Viterbi algorithm, [6], can be used to determine the most likely set of fixed edges in a very fast manner, providing the boundary locations for symbol S_i. We denote the left and right edge locations for symbol S_i as $E^-_{S_i}$ and $E^+_{S_i}$.

3.3 Getting Digit Estimates

After detecting symbol boundaries $\{E^{\pm}_{S_i}\}_{i=1}^{12}$, a simple inner product of the barcode strip for each symbol with the patterns for all the digits is used to estimate the likelihood that a symbol codes for a particular digit. We denote the digit encoded by symbol i as s_i. Note that it is guaranteed that an alternate polarity band of width at least Δ is present at each end of each symbol. To increase the robustness of the inner product, the binary pattern for a particular digit is padded on each side with an alternate polarity band of size Δ, giving extended digit patterns. These patterns are used to reduce errors due to erroneous detection of fixed edges. For each symbol i and each digit d, the inner product gives a likelihood

$$L(s_i = d) = \langle \boldsymbol{\beta}_{E^-_{S_i}:E^+_{S_i}}, \boldsymbol{\delta}_d \rangle, \quad \forall i = 1, 2, \ldots, 12; \quad d = 0, 1, \ldots, 9, \tag{18}$$

where we use the notation $\boldsymbol{\beta}_{i:j}$ to denote a slice of vector $\boldsymbol{\beta}$ containing elements $\beta_i, \ldots, \beta_j$, and $\boldsymbol{\delta}_d$ is an ideal symbol pattern that encodes the extended digit d, e.g., for the digit $d = 0$, it is an alternating pattern of Δ 1's, followed by 3Δ 1's, 2Δ -1's, Δ 1's, Δ -1's, and Δ 1's, as shown in Figure 7. Since the pattern is binary, it is straightforward to use the antiderivative of the barcode scanline to speed up the evaluation of the inner products, so that each inner product can be calculated by six summations.

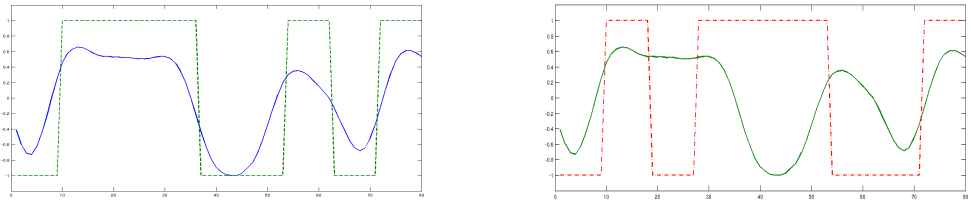

Figure 7: An example of the inner product approach to estimate barcode digits. The intensity pattern is given by the solid line for the part of the barcode strip corresponding to a symbol, and the dashed line indicates the ideal pattern corresponding to a given digit. In this case, the actual symbol is a 0. The ideal patterns for digit 0 ($\boldsymbol{\delta}_0$, correct) is shown on the left, and the ideal pattern for digit 4, ($\boldsymbol{\delta}_4$, incorrect) is shown on the right.

At this stage, most algorithms only use the most likely digits and do a parity check to verify whether the detection is accurate. Instead, in Bayesian fashion, we use a joint estimation of all 12 digits such that the parity equation (12) is satisfied.

3.4 Joint Barcode Estimation

To decode the barcode, the most likely set of symbols that satisfy the parity equation (12) need to be determined, i.e.,

$$\mathbf{s}_{\text{opt}} = \arg\max_{\mathbf{s}} \sum_{i=1}^{12} L(s_i) \quad | \quad \left(3 \times \sum_{i \text{ odd}} s_i + \sum_{i \text{ even}} s_i \right) \mod 10 = 0. \tag{19}$$

However, since the digits are not independent, it is not straightforward to find the solution to (19). As such, to correctly find the joint probability estimate, a set of new auxiliary random variables $c_i \in \{0, \ldots, 9\}$ where $i = 1, \ldots, 12$, corresponding to a running parity digit, are used. Let

$$c_i = \pi_i(c_{i-1}, s_i) = \begin{cases} 3s_1 \mod 10, & \text{if } i = 1 \\ (3s_i + c_{i-1}) \mod 10 & \text{if } i \text{ is odd} \\ (s_i + c_{i-1}) \mod 10 & \text{if } i \text{ is even} \end{cases} \tag{20}$$

Note that s_i is also uniquely determined from c_i and c_{i-1}. Let π_i^{-1} be this reverse mapping for symbol i, i.e., $s_i = \pi_i^{-1}(c_{i-1}, c_i)$. Using these auxiliary variables that form a Markov chain, the parity constraint is changed to the constraint that $c_{12} = 0$. We can then write

$$\Pr\{\mathbf{c}(\mathbf{s})\} = \Pr\{c_1\}\Pr\{c_2|c_1\} \cdots \Pr\{c_{12}|c_{11}\} = \Pr\{s_1\}\Pr\{s_2\} \cdots \Pr\{s_{12}\} \tag{21}$$

or

$$L(\mathbf{c}(\mathbf{s})) = L(c_1) + L(c_2|c_1) + \ldots + L(c_{12}|c_{11}) \tag{22}$$

where $L(c_i|c_{i-1}) = L(\pi_i^{-1}(c_{i-1}, c_i))$, and is given by (18).

The most likely sequence is then calculated using the Viterbi algorithm, [6], starting from an end state of 0. Once the most likely $\mathbf{c}$ is determined, $\mathbf{s}_{\text{opt}}$ is calculated. To make sure that this estimate is reliable, two additional checks are performed:

1. A modified Viterbi algorithm is used to also calculate the second most likely sequence. If the most likely sequence is not significantly better than the second most likely sequence (as determined by a threshold on the energies of the two sequences), the estimate is considered untrustworthy. On other words, we require that $L(\mathbf{s}_{\text{opt}}) > L(\mathbf{s}_{\text{2nd}}) + \tau_{\text{likelihood}}$.

2. If the estimate passes the first test, it is allowed to be at most 1 digit different from the individually most likely estimates for each symbol, i.e., it is required that

$$d_{\text{Hamming}}(\mathbf{s}_{\text{opt}}, \mathbf{s}_{\text{ind}}) \leq 1, \text{ where } \mathbf{s}_{\text{ind}} = \{(s_1, \ldots, s_{12}) \mid s_i = \arg\max_d L(s_i = d), \forall i = 1, \ldots, 12.\} \tag{23}$$

   where $d_{\text{Hamming}}(\mathbf{x}, \mathbf{y})$ is the Hamming distance, defined as the number of positions where two strings $\mathbf{x}$ and $\mathbf{y}$ of equal length differ.

An estimate is considered reliable only if it passes both tests. Otherwise, it is discarded, and the algorithm waits for a new frame.

References

[1] E. Tekin, J. M. Coughlan, and D. Vasquez, "S-K smartphone barcode reader for the blind," *Journal on Technology and Persons with Disabilities*, 2013.

[2] E. Tekin and J. M. Coughlan, "A mobile phone application enabling visually impaired users to find and read product barcodes," in *Proc. 12th International Conference on Computers Helping People with Special Needs*, 2010.

[3] H. Scharr, "Optimal operators in digital image processing," Ph.D. dissertation, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Aug. 2000.

[4] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer, Feb 2009. [Online]. Available: http://www-stat.stanford.edu/~tibs/ElemStatLearn

[5] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 790–799, Aug. 2005.

[6] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. 13, no. 2, pp. 260–269, Apr. 1967. [Online]. Available: http://dx.doi.org/10.1109/TIT.1967.1054010