



## **A Visually Significant Two Dimensional Barcode**

Doron Shaked, Avi Levy, Zachi Baharav<sup>1</sup>, Jonathan Yen<sup>2</sup>

HP Laboratories Israel<sup>3</sup>

HPL-2000-164 (R.1)

December 14<sup>th</sup>, 2001\*

E-mail: [dorons@hpli.hpl.hp.com](mailto:dorons@hpli.hpl.hp.com), [avi@hpli.hpl.hp.com](mailto:avi@hpli.hpl.hp.com)

2-D Barcode,  
information  
hiding,  
watermarking

2-D barcodes are two-dimensional graphical patterns that encode information. They allow for higher information density than the standard one-dimensional barcodes, but usually have an unpleasant appearance. This report describes a system, consisting of an encoder and a decoder, producing a Visually Significant 2-D Barcode (VSB). The VSB encodes information in a 2-D pattern that is visually similar to a pre-specified 2-D gray scale image. It enjoys the high information density typical to 2-D barcodes but avoids their unpleasant appearance.

A note about this document: This report is a compilation of a few separated documents, each describing a different aspect of the VSB system. The reader may notice that different sections were written separately, and often for different purposes. The purpose of this report is to provide a reference for details that are out of scope in subsequent VSB related publications.

\* Internal Accession Date Only

Approved for External Publication?

<sup>1</sup> Currently with Agilent Technologies

<sup>2</sup> Currently with the HPL - Imaging System Laboratories in Palo Alto

<sup>3</sup> HP Labs – Israel, Technion City, Haifa 32000, Israel

© Copyright Hewlett-Packard Company 2002

# 1. Introduction

## 1.1 Background:

Barcodes are information carrying graphical patterns designed for easy and reliable automatic retrieval. The most common barcodes are known as *one-dimensional barcodes*. These graphical patterns vary in a single dimension (e.g. horizontal), and are constant in the other (vertical) dimension. In order to convey more information on the same surface area, the constancy in the vertical dimension has to be abandoned for more intricate patterns, known as *two-dimensional barcodes*.

One-dimensional barcodes are employed in low information content applications like product index registry (e.g. automatic price tagging and inventory management), or serial number registry (e.g. test-tube tagging in automated medical tests). Two-dimensional barcodes are used in applications that require more information contents like mail addresses (for automated mail reading and distribution systems), or compressed content of a printed page (to avoid the need for optical character recognition).

Two-dimensional barcodes are graphical patterns composed usually of dots. They are rendered using two-toned dots (e.g. black dots on a white background), and occupy, usually a rectangular area. Two-dimensional barcodes incorporate various registration and fiducial marks, enabling automated identification, and accurate registration of the barcode, which might be read-in in arbitrary orientations. In addition, two dimensional barcode systems employ various error correcting codes for reliable automated retrieval.

## 1.2 What is new:

We propose a two-dimensional barcode system that has any, or all, of the following new features in addition to the above mentioned features:

- The barcode pattern has some visual significance. In contrast to current patterns the proposed system uses patterns, which make sense as graphical entities. For example:
  - ◆ Logo, like a company, application, or action logo.
  - ◆ Graphics, like frames, button marks, or background.
  - ◆ Text boxes, like a box reading “This box contains important data!” containing, naturally the important data itself – as a barcode embedded in the graphical design of the text, and or the background.
  - ◆ Images.
- The barcode pattern is robust to:
  1. Consecutive photocopying by common office copiers (analog and digital copiers).
  2. Common office document degradations like folds, stains, marks, and staples.
- The barcode is printed and read by standard office equipment like printers, scanners, copiers, and multi-functional equipment.

## 1.3 Why is it important:

In many consumer applications, the current visually meaningless barcode patterns are prohibitive; since users are more likely to decline the benefits of the application than put “a barcode” on their letterhead. In that sense, the proposed barcode is aimed at changing the sentimental value attached to the use of barcodes, and move its context from the impersonal commercial and industrial setting to the business and even the home environments. Different applications may easily incorporate different logos or other barcode graphic, thereby enabling a customized personalization of the barcode.

The ability to produce and to read back barcodes using office equipment is critical for many applications, and so is the robustness to photocopying. These requirements impose some very restrictive constraints on the barcode patterns and the barcode reading and decoding methods, to name a few:

- Small dots are prohibited.
- Printer and scanner distortions should be expected and dealt with in the registration and the error correction stages.
- Dot size and shape might have to be pre-compensated to be robust to expected deformations in the photocopy process.
- Registration should align a pattern that underwent transformations more general than translation, and rotation (e.g. affine transformation).

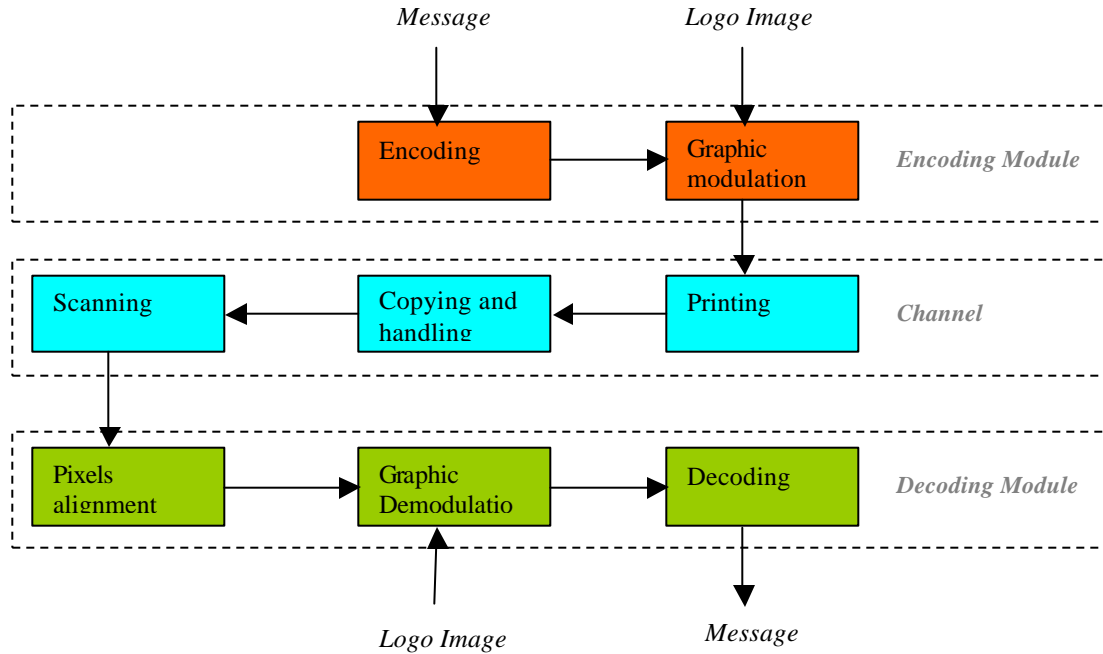
#### 1.4 The organization of this document

This document is a compilation of few separated documents, each describing a different aspect of the VSB system. The reader may notice that different sections were written separately, and often for different purposes. Section 2 presents a general scheme of the encoding and decoding procedures of the VSB. In section 3 the current implementation is detailed. Section 4 elaborates on the XOR BarCode modulation scheme, which is the method by which the information is encoded in 2-D patterns. In section 5 we details the correction of geometric pattern deformations procedure, which is a preprocessing stage that proceed the decoding stage. Section 6 describes the VSB decoding procedure. The report is ended with an appendix that analyses some theoretical aspects of visual significant 2-D barcodes. Related documents can be found in the bibliographic section.

## **2. Visually Significant Barcode – Encoding and Decoding Schemes.**

The proposed barcode is composed of two modules. The barcode-encoding module (red blocks in Figure 2.1) processes a message and a predefined *logo-image* (containing a logo, or any other graphical objet), and renders the image as a specific pattern, called the *barcode-pattern*, on a hardcopy. Alternatively, it produces a representation of the barcode pattern, which may be incorporated in a document and printed on a hardcopy. The barcode pattern appears like the logo image, however, it contains a retrievable version of the message. If the hardcopy containing the barcode pattern, or a copy of it, is scanned

and processed by the barcode-decoding module (green blocks in Figure 2.1), the original message will be decoded. The two modules and the expected hardcopy-handling path (cyan blocks labeled as channel) are detailed in Figure 2.1.



**Figure 2.1:** A coarse block diagram of the proposed barcode. The proposed barcode is composed of two modules (barcode encoding, and barcode decoding), communicating through an assumed office-type document-handling path (channel).

## 2.1 Barcode encoding:

The barcode-image is a binary image (black and white dot pattern) produced by the barcode-encoding module as follows:

1. The message is initially compressed into a compact representation.
2. Next it is coded using an error correcting code with an output alphabet of size  $L$ . The error correcting code provides robustness to errors due to degradations introduced in the channel. Codes may be interleaved to protect against burst errors. At the end of this stage the message is encoded in a sequence of  $Q$  symbols over  $\{1, 2, \dots, L\}$ .
3. The logo-image, an  $M \times N$  pixel image is partitioned to a rectangular array of  $K \times K$  pixel sub-images, called *logo-matrices* (for simplicity we assume that  $N$  and  $M$  are multiples of  $K$ ).
4. Some image area (corresponding to  $R$  logo-matrices) is used for predefined fiducial marks.

5. The remaining  $P = M \times N / (K \times K) - R$  logo-matrices are ordered in a sequence. Each logo-matrix in that sequence encodes a single symbol of the coded message sequence. Note that, if  $P < Q$  the message cannot be encoded completely.
6. Logo-matrices are converted to  $K \times K$  binary *barcode-matrices*, using one of a predefined set of  $L$  distinct halftoning algorithms. The choice of the halftoning algorithm is arbitrated by the corresponding symbol in the coded message sequence. Note that the  $L$  halftoning algorithms are preferably designed to:
  - Produce distinctly different barcode-matrices for any legitimate logo-matrix.
  - Produce visually pleasing halftone patterns.
7. The barcode-image is composed of the fiducial marks and the barcode-matrices placed at their corresponding locations. The barcode image is rendered using dots of specific size and shape, designed so as to survive the channel degradations.

## 2.2 Barcode decoding:

The image acquired by the scanner and introduced to the barcode decoding module is a degraded version of the barcode-image. These degradations are attributed to the channel, namely, the printing and scanning processes, and potential office type degradations like copying, stains, folds, staples, and marks. The scanned image is processed as follows:

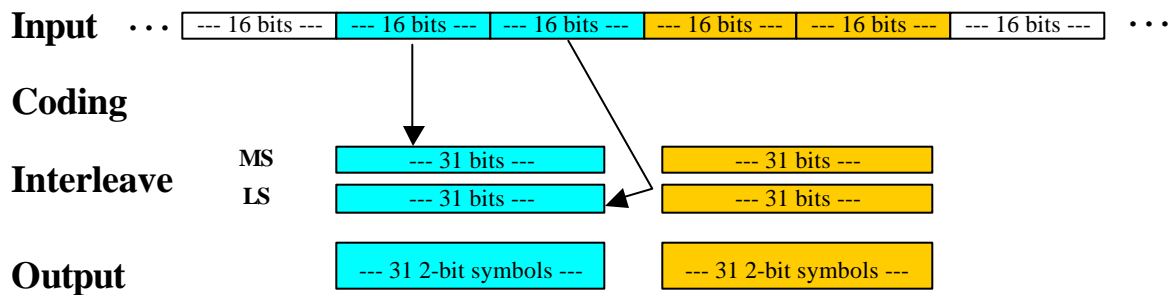
1. Initially the barcode image has to be located on the scanned image (containing usually, the whole page).
2. The fiducial marks are detected. The configuration of these fiducial marks indicates the type of global deformation introduced by the channel
3. Viewing transformation distortions (translation, rotation, and affine) are corrected.
4. The scanned image processed to correct for the degradations due to the printing and scanning channel. It is next partitioned to a rectangular array of sub-images, corresponding each to a single barcode matrix. The scanned image pattern may help to detect some other local or global transformations, which may be corrected at this stage.
5. Each sub-image is compared against the set of  $L$  possible barcode-matrices (outputs of the  $L$  distinct halftoning algorithms, given the corresponding logo-matrix). The best match is selected to represent the sub-image in a sequence of  $P$  symbols over  $\{1, 2, \dots, L\}$ .
6. Next the (possibly erroneous) sequence, originally coded with an error-correcting code is decoded, thereby eliminating the effect of possible errors due to the degradations of the channel.
7. The sequence is decompressed to give the original message.

### 3. Visually Significant Barcode Implementation

The above description of the proposed barcode is a general description. In this section the current implementation is detailed in enumerated items corresponding to the items in the previous section.

#### 3.1 Barcode encoding:

1. Currently we do not implement any source compression. Any off-the-shelf compression algorithm may fit in, though for some target implementations as URLs a tailored compression scheme would be preferred.
2. We currently use a standard  $16 \rightarrow 31$  bit BCH code correcting for 3 errors. In the current implementation  $L = 4$  (2 bits). To ensure that each erroneously detected symbol induces a single error in two different code words, rather than inducing two errors in a single code word, the MS bits and the LS bits are coded separately and interleaved, see Figure 3.2.



**Figure 3.2:** The error correction code. Consecutive 16 bit batches are coded to 31 bit code words. Code words are multiplexed to MS or LS symbol bits.

3. In the current implementation  $K = 2$  (larger  $K$  is a promising alternative). The size of the logo image ( $N$  and  $M$ ) should therefore be even. For simplicity let  $N = 80$  and  $M = 40$ , however, the exact values are a free parameter of the logo designer (as is the logo image itself).
4. We use the Four Corners of the image for fiducial marks. In each corner we take an area of  $4 \times 4$  pixels ( $2 \times 2$  matrices). The fiducial marks are as follows: The whole area is rendered white, except for the extreme location in each corner, which are rendered black (e.g. the upper-left pixel – for the upper-left fiducial mark).
  - We chose this pattern, since in our channel model dots may be blurred or move relative to each other. This pattern makes sure that the black fiducial dots do not merge with neighboring dots, and stand out clearly on white background.
  - Other fiducial patterns might be considered.
5. The remaining logo-matrices (784 for our choice of  $N$  and  $M$ ) are ordered in raster scan. More sophisticated interleaving methods are also considered. Those can provide

robustness to burst-type degradations expected from stains, marks, or systematic printer/scanner distortions.

The 784 logo-matrices can accommodate slightly more than 25 batches of 31 matrices ( $Q = 25 \times 31 = 775$ ). Each batch codes 2 batches of 16 input bits (one for the MS bits, and another for the LS bits). Thus for the specified parameters the barcode may encode  $25 \times 16 \times 2 = 800$  bits of information.

6. There are many halftone methodologies from which one can choose the  $L = 4$  distinct halftoning algorithms. In the current implementation we chose to use a variation of the old, fixed-half-tone-pattern halftoning method, however we consider using dithering which may eliminate some of the following limitations:

- The logo image is limited to be a 2-tone image.
- If black is 0, and white is 1, the bright tone  $b$ , and the dark tone  $d$ , are such that  $d = 1 - b$ .
- The  $L$  halftoning algorithms correspond to  $L$  distinct  $K \times K$  pattern-matrices. Each of which contains  $b \cdot K \times K$  black dots on white background – notice that this constitutes another limitation on  $b$ .

Given a logo-matrix, and the selected pattern-matrix, the resulting barcode-matrix contains the pattern matrix values in the places corresponding to the bright pixels in the logo-matrix, and their complementary otherwise. In our implementation we chose the  $L = 4$ ,  $2 \times 2$  matrices of Figure 3.3, with  $b = 0.25$ .



**Figure 3.3:** *The four pattern-matrices used for graphical*

7. In the current implementation the dots are simply rendered as square dots at 85 dots-per-inch (dpi). Larger dots are more robust to channel degradations, and smaller dots enable more information on the same area of the paper. 85dpi is the smallest dot size for which we can assure acceptable error resilience. Figure 3.4b is an example of the proposed barcode visualizing the logo-image in Figure 3.4a.



**Figure 3.4:** *Logo-image in a, and corresponding barcode in b.*

Note that it is possible to vary the size of the dots:

- According to intensity, e.g. black dots should be slightly larger than white dots.
- According to neighborhood, e.g. minority colored dots should be larger.

It is also possible to change the shape of the dots. For example hexagonal dots on an hexagonal grid are likely to be more robust to channel degradations, and thus be rendered at smaller dpi.

## 3.2 Barcode decoding:

1. Currently we do not implement any barcode location procedure; the barcodes are located roughly at the same location in the scanned image, surrounded by white pixels. We refer to that location as the barcode-zone.
2. In the current implementation the barcode-zone is scanned in a zigzag scan from all the Four Corners. Figure 3.5 depicts one such scan (marked by cyan) at the upper-left corner. The first dark pixel of each scan (requires a threshold) is considered to be a part of the corresponding fiducial pixel, and is used as an anchor pixel for that mark (red dot in Figure 3.5). A standard flood-fill algorithm locates all the dark pixels connected to these anchors (green outline in Figure 3.5). The centers of the fiducial marks are then computed as the average (center of mass) of the pixels of each mark (blue dot in Figure 3.5). This zigzag scan enables a robust detection of the anchor points even in the presence of significant rotations.

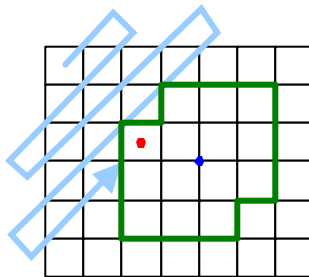


Figure 3.5: The zigzag scan order (cyan), the anchor point (red), the set of connected dark points (green), and their center of mass (blue).

3. Initially we set the center of our coordinate system 10 pixels above and to the left of the center of the upper-left fiducial mark. That eliminates the translation problem. Next we find the relative rotation of the center of the upper-right fiducial mark, and rotate the image back. Last we find the skew factor as the horizontal translation of the centers of the lower marks relative to the centers of the upper mark. If a skew is detected, the image is corrected also for the detected skew. The transformation procedures are standard procedures in computer vision and image processing algorithms. We use bilinear interpolation. Note that:
  - It is possible to use many other interpolators here.
  - It is possible to correct for both the rotation and the skew in a single transformation,
  - The 4 fiducial marks enable correction of global transformations with up to 8 degrees of freedom. In the current implementation we use only 4 (2 - translation, 1 - rotation, and 1- skew).
4. The previous step results in a rectangular image. We can measure it and slice it to  $20 \times 40$  rectangular sub-images (in our case  $M/K = 20$ , and  $N/K = 40$ ). Significant improvements are gained when this simple procedure is replaced by the deformation correction procedures detailed in section 5.



5. Considering the simple halftone patterns used, and the fact that we further limit the logo design to have a constant brightness in every logo-matrix, we chose to apply 4 simple vector products to every sub-image. The 4 vectors are Gaussian profiles, centered each at the center of a quadrant of the sub-image. For dark sub-images, the pattern (see Figure 3.3) corresponds to the quadrant having the maximal value, and for bright sub-images, the pattern corresponds to the quadrant having the minimal value.  
 In the more general case, it is possible to apply any maximum-likelihood-type of detector to determine, which of the  $L$  possible halftones is the most likely to have produced the corresponding sub-image.
6. The corresponding standard BCH error correction is performed followed by the appropriate reordering of the bits (see Figure 3.2).
7. Since we do not use compression, no decompression is necessary.

## 4. XOR BarCode Modulation

The modulation stage for the Barcode is the stage where binary information (usually coded for error-correction and such), turns into an image. This has to be done considering two requirements:











1. The information should be retrievable from the image
2. The resulting image should render the input image (it should look like it).

For that purpose it is important to note that the input image is made of pixels whose gray values are limited to  $g$  and  $1-g$ .

In the previous barcode modulation scheme we rendered each pixel from the input image, by expanding it to one of four different  $2 \times 2$  dot patterns. Patterns rendering bright pixels ( $g=0.25$  gray value) were composed of 3 white dots and a single black dot, and vice versa for dark pixels. Note that the two modulation requirements were naturally fulfilled:

1. The location of the minority dot codes 2 bits of information.
2. Average brightness of output patterns is identical to the value of the respective inputs.

The table below depicts the  $2 \times 2$  dot patterns as a function of the input pixel and the code.


























Input	Code			
	0	1	2	3
				
				

In the new barcode modulation method (XOR modulation) we take  $N \times N$  input pixel patterns, and render them as  $N \times N$  dot patterns. The rendering patterns are the XOR between the input pattern and the relevant code pattern. Let us detail the method for the simple case of  $N=2$ , and for the actual implementation for which  $N=3$ .

For  $N=2$ , we have four code template patterns, the four modulation patterns for a bright input pixel in the previous modulation method (the first line in the table above). The table below depicts  $2 \times 2$  dot pattern corresponding to a few examples of input pixel patterns.

Note that here also the two modulation requirements were fulfilled, in a similar manner to the previous modulation method:

1. The hamming distance between outputs is identical to the distance between the codes. In other words, theoretically, for every  $2 \times 2$  input pattern, the four corresponding output patterns are as different from each other as the code patterns. Since the code patterns are the modulation patterns of the previous method, the information is, theoretically not less retrievable in the new modulation, than in the previous one.
2. In the new modulation method each pixel is rendered as a single dot, thus in the output it is either black or white. However, the probability of bright pixels to be rendered white is 0.75, and the probability of dark pixels to be rendered white is 0.25. Thus on the average the input gray value is maintained.

Input Examples	Code			
	0	1	2	3
				
				
				
				
				

For  $N=3$  we first have to choose  $g$ , and then select an appropriate set of code template patterns. We chose  $g=2/9$ , which enhances the visual quality of the output pattern, since the contrast  $(1-g)-g = 1-2g$ , is larger for  $g=2/9$ , then for  $g=0.25$ . The corresponding set of code template patterns is all the possible combinations of two pixels in a  $3 \times 3$  set, altogether 36 patterns. Some of these patterns are shown in the context of XOR modulation in the following table.

We could have used all the 36 code template patterns efficiently coding  $\log_2 36 = 5.17$  bits in every pattern. In the current implementation we chose however to give up the fractional capacity, and limited ourselves to 32 codes. In order not to be biased by the selection of a predefined set of codes, we keep toggling through the 36 codes as follows: The code  $C_{36}$  (one of 36) is obtained from the desired code  $C_{32}$  (one of 32) by:

$$C_{36} = \text{mod}_{36}(C_{32} + \text{counter})$$

Where the counter is incremented after every use. During decoding  $C_{32}$  is decoded from  $C_{36}$  by:

$$C_{32} = \text{mod}_{36}(C_{36} - \text{counter})$$

		Code					
		0	1	2	3	4	...
Input Examples							

The new modulation method is a natural extension of the previous modulation method. This is obvious from the last two rows in the table, where constant patterns result in patterns identical to previous modulation. Furthermore, it is straightforward to extend the proposed modulation method to more complex  $n \times n$  modulation patterns.

The problem with the new modulation is at the decoding stage. Where previously the code was determined simply by looking for the maximal/minimal average gray value in a dot, now it involves Bayesian estimation (we handled this as well, though for coherence it is not detailed here).

## **5. Correction of Geometric Pattern Deformations**

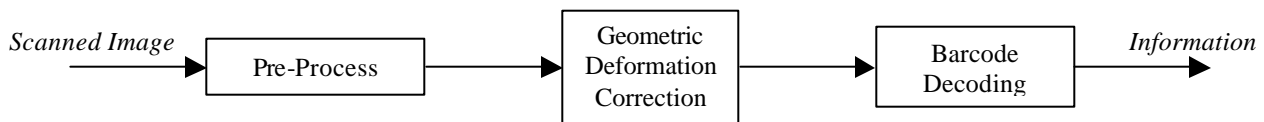
We describe a correction method for VSB binary patterns that have undergone geometric deformations due to office-type degradations including photocopy. Copies of barcode dot patterns are deformed as follows:

1. Shape deformations are responsible for the fact that black dots change their size and either shrink or expand. This deformation depends usually on the brightness setting of the copier. In a dark setting black dots expand, and in bright settings they shrink.
2. Space deformations are responsible for the fact that dots corresponding to certain coordinates in the original image are located at different coordinates in the copy. Global space deformations like translation, rotation, and affine transformation are corrected using the registration marks. However, additional local deformations occur which are harder to characterize and correct for. Most deformations due to copying are approximately separable, namely, copies of coordinates  $(x_0, \cdot)$  in the original are located at coordinates  $(x_0 + \Delta, \cdot)$ , and likewise  $(\cdot, y_0)$  are mapped to  $(\cdot, y_0 + \Delta)$ .

The correction is composed of two stages. In the first we use morphological operations to correct for shape deformations, and in the second we use row/column gradient statistics to correct for local approximately separable space deformations.

### **5.1 General Scheme of Geometric Pattern Correction**

The proposed correction module for geometric pattern deformations is located between a pre-process module and a decoding module, as in Figure 5.1.

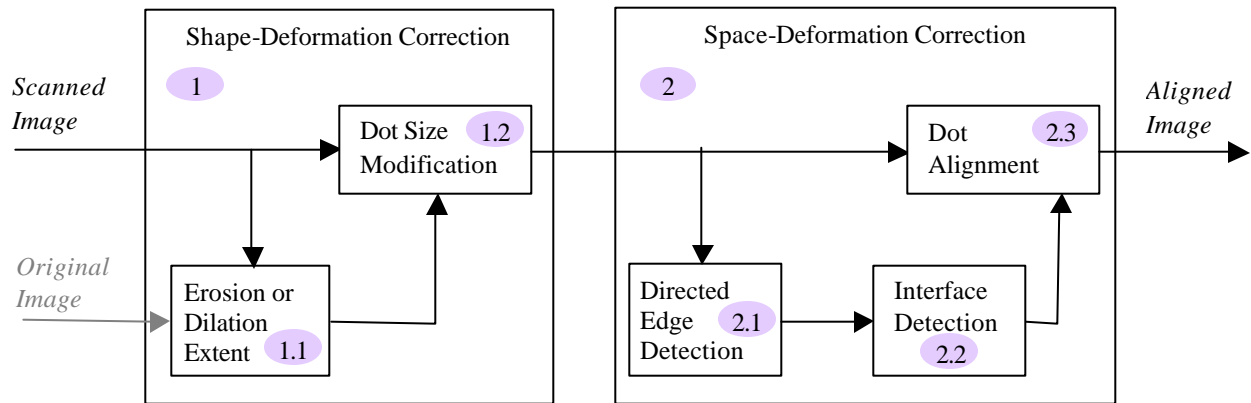


**Figure 5.1:** A coarse block diagram of a barcode decoding system in which the proposed geometric deformation correction module may be located.

The correction module is composed of two main parts, described below, and in Figure 5.2:

1. A correction module for shape deformations is composed of:
  - 1.1. A module for determining whether black dots have eroded or expanded relative to the original pattern, and if so, to what extent. The extent is quantified as a real number standing for the radius of erosion or dilation. This module may be implemented in several ways, for example:
    - 1.1.a. Analyzing average darkness of the pattern, and comparing it to the average darkness in the original pattern, and translating the darkness differences to erosion or dilation radius. This method requires the availability of the average darkness of the original image, a reasonable assumption in some applications (including ours).
    - 1.1.b. Analyzing dot shapes, locating single dots in the pattern, and comparing their radius to the radius of single dots in the original barcode pattern. This method requires the availability of the original dot radius, a reasonable assumption in all applications.
  - 1.2. A module for shrinking or expanding black dots to compensate for the respective expansion or shrinkage of the pattern due to the shape deformation. This module may be implemented in several ways, for example:
    - 1.2.a. Morphological operations eroding or dilating the black pattern at the specified radius.
    - 1.2.b. Modifying the threshold defining the black pattern in the scanned image.
2. A correction module for local space deformations determines the dot grid in the scanned image by locating the interfaces of dot-rows and dot-columns. This module is composed of:
  - 2.1. A module for detecting horizontal and vertical edges. Horizontal edges are used for locating dot-column interfaces, and vertical edges for dot-row interfaces. This module may be implemented in several ways, for example:
    - 2.1.a. Directed (horizontal and vertical) gradient estimation.
    - 2.1.b. Zero crossing of the directed (horizontal and vertical) Laplacian.
  - 2.2. A module for locating dot-row and dot-column interfaces from the edge information. This module may be designed to correct either separable space deformations or general ones, and may be implemented in several ways, for example:
    - 2.2.a. To correct for separable space deformations the absolute value of the horizontal gradient may be summed in columns. A large value is an indication of dot-column interface. And vice versa for row interfaces.
    - 2.2.b. To correct for general space deformations column interfaces may be estimated row by row requiring consistency with: gradient magnitudes, near by interface locations, and interface locations in previous rows. And vice versa for row interfaces.
  - 2.3. A dot alignment module, correcting for the space deformations according to the dot interfaces, which may be implemented in several ways, for example:
    - 2.3.a. Augmenting the image with a list of true dot centers.

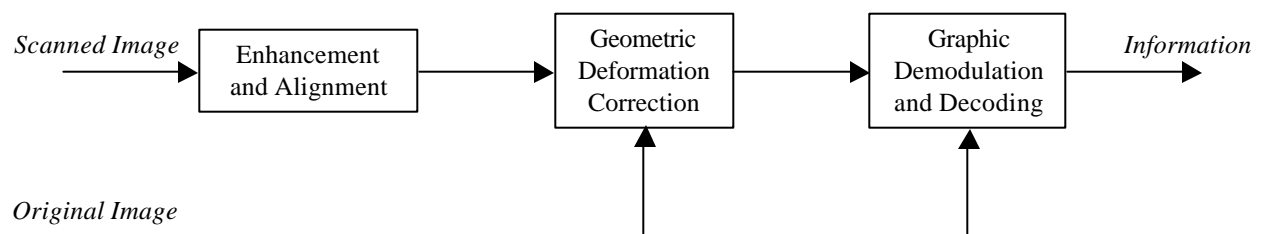
2.3.b. Virtually moving the dots into their original location by composing a new image made of sub-images cropped around warped dot centers.



**Figure 5.2:** A block diagram of the proposed geometric deformation correction module.

## 5.2 Current implementation of Geometric Pattern Correction

The Barcode decoding system, for which the proposed pattern correction was designed and built, reads in a scanned document containing the image of a *known* visually significant 2D Barcode. It first enhances the scanned image, locates the synchronization marks, and uses them to correct for global space transformations (translation, rotation, and shear). In the next stage it corrects for the other geometric deformations, which are the subject of this subsection. Subsequently it analyzes the barcode pattern (graphic demodulation), and decodes it to give the embedded information, see Figure 5.3.



**Figure 5.3:** A coarse block diagram of the barcode decoding system in which the preferred implementation of the geometric deformation correction module is located.

The preferred implementation is composed of the (a) implementations of the previous section. Next, the various implementations will be detailed. The (a) implementations fit in well with each other. Other system combinations may require some fitting.

### 5.3.1 Details of Erosion/Dilation Extent Determination Module (1.1.a)

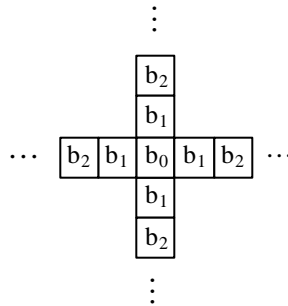
In the preferred system the relative area of the black pattern is equal to the average gray value of the original image (the image which the barcode renders). This is a reasonable assumption for general visually significant barcodes. Therefore, the scanned image is first binarized using a threshold function and the relative area,  $b$ , of the white part is compared to the average gray value,  $g$ , of the original image. If  $b$  is smaller than  $g$ , the required morphological operation is dilation of the black dots, (erosion if larger). The radius,  $r$ , of the required morphological correction is a function of the absolute difference  $|g - b|$ . For our implementation we approximated this function as a linear function:  $r = 9 \cdot |g - b|$ , where  $b$  and  $g$  are represented as fractions in the range  $[0, 1]$ .

### 5.3.2 Details of Erosion/Dilation Extent Determination Module (1.1.b)

For another possibility to implement this module one first applies super-resolution edge-detection on the deformed dot shapes, after which horizontal and vertical black runs are measured. Runs originating in  $n$  dots are measured  $n \cdot R + 2r$ , where  $R$  is the dot radius, and  $r$  the deformation radius. The deformation radius  $r$  is the one minimizing the best robust square fit of the measurements to the above model. For a similar algorithm look in [1].

### 5.3.3 Details of Dot-Size Modification Module (1.2.a)

In the preferred system the deformations are best modeled as a morphological dilation or erosion of the black pattern, which are best corrected for by erosion or dilation respectively. In the preferred implementation the structuring element is a cross shape with entries:  $b_0$  in the center,  $b_1$  in coordinates  $(\pm 1, 0)$  and  $(0, \pm 1)$ ,  $b_2$  in coordinates  $(\pm 2, 0)$  and  $(0, \pm 2)$ , and so on, as in Figure 5.4.



**Figure 5.4:** *The preferred morphological structuring*

The corrected image is the morphological gray scale dilation or erosion of the image with the above structuring element. Namely, one places the structuring element on every pixel in the input image. The value of the dilation at the corresponding pixel is the maximum of the differences of neighborhood pixel values with corresponding structuring-element values. For erosion one simply takes the minimum of sums:

$$Dilate_{i,j} = \mathbf{Max}_{k,l \in SE} \{ I_{i+k,j+l} - S_{k,l} \} \quad \text{and} \quad Erode_{i,j} = \mathbf{Min}_{k,l \in SE} \{ I_{i+k,j+l} + S_{k,l} \}$$

where  $SE$  is the set of valid structuring-element coordinates, and  $I_{m,n}$ ,  $S_{m,n}$  are image and structuring-element values at coordinates  $(m,n)$ .

The values  $b_0, b_1, b_2, \dots$  should be such that they perform as a structuring element of a given radius. For integer-valued radii this is simple:

$$\mathbf{r} = i \Rightarrow b_j = \begin{cases} 0 & \text{for } j \leq i \\ 1 & \text{for } j > i \end{cases}$$

For non-integer radii we used the following transformation:

$$\mathbf{r} \in [i-1, i] \Rightarrow b_j = \begin{cases} 0 & \text{for } j < i \\ 0.22 \cdot (i - \mathbf{r}) & \text{for } j = i \\ 1 & \text{for } j > i \end{cases}$$

which was found to give a linear correction in terms of deformation radius as measured by module 1.1.a. Namely, if an image with a measured deformation radius of  $r_0$ , is corrected with radius  $-r_1$ , the subsequent deformation radius will measure  $r_0 - r_1$ .

#### 5.3.4 Details of Dot-Size Modification Module (1.2.b)

Another possibility to implement this module is simply to modify the threshold defining the black pattern. Since the black pattern is obtained from the scanned image by a threshold, one can modify it thereby modifying the area of the black pattern up to the required ratio. However, in our implementation we found this simple correction to be not satisfactory, probably because the morphological deformation model was more accurate.

#### 5.3.5 Details of Directed Edge Detection Module (2.1.a)

In the preferred implementation we simply used forward derivative in the horizontal or vertical direction to estimate the directed edges in the respective direction. One may use equivalently any directed edge detection kernel.

#### 5.3.6 Details of Directed Edge Detection Module (2.1.b)

Another possibility to implement this module is to use zero crossing.



### 5.3.7 Details of Interface Detection Module (2.2.a)

In the preferred system the barcode covers a relatively small area on the paper. In such applications one can safely approximate the space deformation to be separable. This means that the row and column interfaces are aligned with pixel rows and columns, and the deformation is expressed only in their uneven distribution.

At column interfaces one can expect to see many transients between black dots on the right of the interface and white dots on the left, or vice versa. We therefore sum up the absolute values of horizontal gradients in columns and determine the interface at columns with high peaks of gradient figure. To find row interfaces one can transpose the image and do the same (or transpose the above procedure).

In the preferred implementation we also provide for outlier interfaces, in which for some reason there was a weak gradient activity (e.g. in most rows dots on both sides of the interface had identical value). This is possible by determining a range (measured from the last interface) in which to look for the new interface. If a no interface is detected in that range, it is determined to be a standard dot-size away from the last interface.

### 5.3.8 Details of Interface Detection Module (2.2.b)

Another possibility to implement this module, which is necessary in applications where the space deformation is distinctly not separable, is the following procedure described for column interfaces. Column interfaces are estimated row by row. In every row the location of each column interface is determined so as to satisfy a few, potentially conflicting consistency requirements. The interface should preferably agree with local large gradient magnitudes, it should not deviate much from its location in the previous row, and finally it should form a quasi-uniform pattern with near by interface locations in the same row.

All this can be implemented as follows (we used it in our watermark demo). Interfaces are recorded in sub-pixel accuracy. Binary gradients are determined by thresholding the scanned image. If a gradient is located within 1.5 pixels from an interface location in the previous row it is associated with that interface. Interfaces with no gradient association keep their location from the previous row. Interfaces with multiple gradient associations relate only to the closest, and determine their new location as a weighted average between its location (weight=0.3), and their location in the previous line (weight 0.7). When all the interfaces for a row have been determined this way, their final location is a weighted average between these locations and the average location of their respective neighbors on the left and right (average extent may range up to several interfaces on each side).

### 5.3.9 Details of Dot Alignment Module (2.3.a)

In the preferred implementation, dots are only virtually aligned by augmenting the scanned image with a list (describing a potentially non-uniform square grid) of dot centers. Dot center coordinates may be computed as either the center of the interface

coordinates on both sides (for separable deformations), or the center-of-mass of the dot for non-separable deformations.

### 5.3.10 Details of Dot Alignment Module (2.3.b)

Another possibility to implement this module is to actually compose a new image from sub-images cropped around located dot centers. Dot centers in the aligned image are thus located on a uniform square grid, each dot covering a square patch of pixels around it. This square patch is cropped out of the respective dot location in the scanned image. This way there are pixels in the scanned image that will not be found in the aligned image, and other pixels that will be copied to several locations in it.

## **6. VSB Decoding Procedure**

For the VSB code to be robust under printing, photocopying and scanning each pixel of the halftoned image  $B$  is duplicated and printed as an rectangular area of  $t \times t$  printed pixels ( $p$ -pixels). These areas are homogeneously composed of either black  $p$ -pixels (for black original pixel) or white  $p$ -pixels (for white original pixel). The printed image is going through consecutive photocopying, after which it is scanned in high resolution.  $S$  denotes the scanned image. Suppose that the scanning resolution is such that each original pixel is now represented by an array of  $r \times r$  scanned pixels ( $s$ -pixels). Due to visual degradation caused by the printing and scanning processes, the resulting arrays of  $s$ -pixels in  $S$  are no longer homogeneous but contain  $s$ -pixels of different gray levels.

In order to decode the message  $M$  from the halftoned image  $B$  the following steps are required:

- (a) Align the scanned image  $S$ .
- (b) For each pixel in the halftoned image  $B$  - identify the borders of the corresponding  $s$ -pixel array in the scanned image  $S$ .
- (c) For each  $s$ -pixel array - estimate the probability that this array have been originated from a black (white) pixel.
- (d) Based on these probabilities, decide for each sub-image of  $(n \times n)$  pixels in the original image  $I$ , which of the code words was used for its halftoning.
- (e) Having identified the code words for each sub-image, the message  $M$  can now be decoded using an error correcting decoding.

Steps (a), (b) and a preprocessing for (c) are described in section 5 and step (e) is considered in section 3.

Assume now that for each pixel in  $B$ , the corresponding  $s$ -pixel array in  $S$  has been identified. On this basis we describe in detail steps (c) and (d) above.

## 6.1 Estimating the probability for a pixel to be white or black

### 6.1.1 Discrimination function

Let  $s$  be an array of  $s$ -pixels corresponding to one original pixel and denote the gray levels of its  $s$ -pixels by  $s_{i,j}$  where  $i,j=1,\dots,r$ . We are looking for a function of the  $s_{i,j}$  that discriminates between arrays that corresponds to black original pixels and those corresponding to white original pixels. We found experimentally that a good linear discrimination function is of the form  $f(s) = \sum_{i,j=1,r} c_{i,j} s_{i,j}$  where the coefficients  $c_{i,j}$  are

taken as grid values of a two dimensional gaussian shaped surface. More explicitly -  $c_{i,j} = \max(c, \exp\{-((i-r/2)^2 + (j-r/2)^2)/\sigma\})$  where  $c$  and  $\sigma$  are constants.

A histogram of the discrimination function computed for each pixel in a scanned half-toned image is demonstrated in figure 6.1. The right side population belongs to “white” pixels and the left side population belongs to “black” pixels.

### 6.1.2 Estimating probabilistic model parameters

Based on the shape of such histograms we modeled the distribution of  $f(s)$  by a bi-modal non-symmetric exponential distributions. More explicitly, we assume that for black (white) pixels  $y=f(s)$  is distributed according to:

$$P(y | \mathbf{a}_L, \mathbf{a}_R, \mathbf{m}) = \begin{cases} \mathbf{a}_L \exp\{-\mathbf{a}_L(\mathbf{m} - y)\} & y < \mathbf{m} \\ \mathbf{a}_R \exp\{-\mathbf{a}_R(y - \mathbf{m})\} & y > \mathbf{m} \end{cases}$$

Since the parameters  $\alpha_L$ ,  $\alpha_R$  and  $\mu$  are different for the “black” pixels and the “white” pixels populations, the overall number of parameters to be estimated in this model is 6. To simplify notations we denote black pixels by 0 and white pixels by 1 and the parameters are index correspondingly -  $\alpha_{0L}$ ,  $\alpha_{1L}$  etc.

In order to reduced the computational burden, we divide the estimation procedure into two steps:

(a) Estimating  $\mu_0$  and  $\mu_1$ .

We partition the observations  $y_i=f(g_i)$  into two populations, using the known expected number of “black” pixels -  $N_0$ , and “white” pixels -  $N_1$ . The “black” population comprises of the smallest  $N_0$  observations and the “white” populations comprises of the rest (biggest  $N_1$ ). For each of these populations we estimate its mode  $\mu$  by the following ML approximated method:

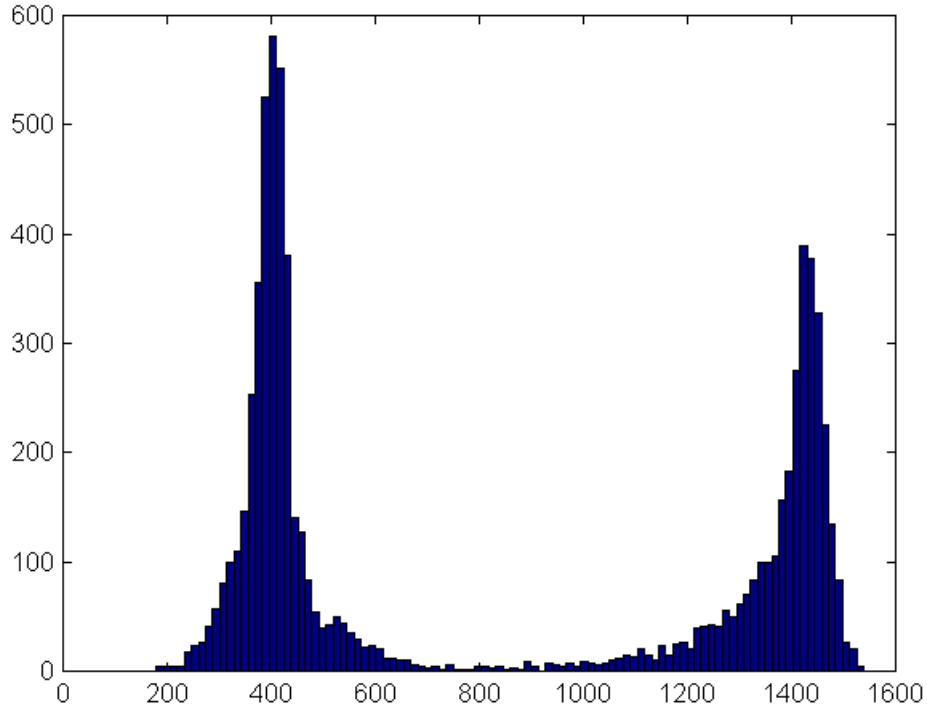


figure 6.1

We look for the value of  $\mu$  (for the sake of simplicity we ignore the 0/1 subscript) that maximizes the ML estimator for  $\mu$ :

$$(2.1) \quad \bar{\mathbf{m}} = \text{Arg}\left\{\max\left[\sum_{i \in I_L} \log \mathbf{a}_L - \mathbf{a}_L(\mathbf{m} - y_i) + \sum_{i \in I_R} \log \mathbf{a}_R - \mathbf{a}_R(y_i - \mathbf{m})\right]\right\}.$$

To find the value that yield the maximum we enumerate all the averages of two consecutive observations in the given populations. Since the true values of  $\alpha_L$  and  $\alpha_R$  are not known we estimate them, for each tested value of  $\mu$  by:

$$(2.2) \quad 1/\mathbf{a}_L = (1/N_L) \sum_{i \in I_L} \mathbf{m} - y_i$$

$$(2.3) \quad (2.3) \quad 1/\mathbf{a}_R = (1/N_R) \sum_{i \in I_R} y_i - \mathbf{m}$$

Where  $I_L$  is the set of observations smaller than  $\mu$ ,  $I_R$  is the set of observations larger than  $\mu$ ,  $N_L = |I_L|$  and  $N_R = |I_R|$ .

Substituting (2.2) and (2.3) in (2.1) and simplifying the expression one gets:

$$\bar{\mathbf{m}} = \text{Arg}\left\{\max\left[N_L(\log N_L - \log \sum_{i \in I_L} (\mathbf{m} - y_i)) + N_R(\log N_R - \log \sum_{i \in I_R} (y_i - \mathbf{m}))\right]\right\}$$

(Note that  $I_L$  and  $I_R$  and hence  $N_L$  and  $N_R$  are both functions of  $\mu$ .)

The estimates for  $\alpha_{0L}$  and  $\alpha_{1R}$  are derived directly from (2.1) and (2.2) using the corresponding estimates for  $\mu_0$  and  $\mu_1$ .

Having estimated  $\mu_0$ ,  $\mu_1$ ,  $\alpha_{0L}$  and  $\alpha_{1R}$  we proceed to estimate the parameters  $\alpha_{1L}$  and  $\alpha_{0R}$  (Since there is no place for confusion we use the notations  $\alpha_1$  and  $\alpha_0$ ). We ignore all observations below  $\mu_0$  and above  $\mu_1$  and define  $I_M = \{y \mid \mathbf{m}_0 \leq y \leq \mathbf{m}_1\}$ ,  $N_M = |I_M|$ . We assume that the observations  $\{y_i : i \in I_M\}$  are samples from a random vector with distribution  $P(y_i, x_i) = \prod_{i \in I_M} P_{x_i} \mathbf{a}_{x_i} \exp\{-\mathbf{a}_{x_i} | y_i - \mathbf{m}_{x_i} | \}$  where  $x_i$  takes the values 0 or 1 with the probabilities  $P_0$  and  $P_1$  respectively. We use the EM paradigm to estimate  $\alpha_1$  and  $\alpha_0$ . The EM algorithm start with initial values for the parameters and perform a series of updating stages that converges to a (local) maximum of the mean ML objective function. To estimate the values of  $\alpha_1$  and  $\alpha_0$  the conditional expectation

$$E[\log P(y_i, x_i) \mid \mathbf{a}_0, \mathbf{a}_1, y_i] = \sum_{i \in I_M} P(0 \mid \mathbf{a}_0, \mathbf{a}_1, y_i) \log(P_0 \mathbf{a}_0 - \mathbf{a}_0 (y_i - \mathbf{m}_0)) + P(1 \mid \mathbf{a}_0, \mathbf{a}_1, y_i) \log(P_1 \mathbf{a}_1 - \mathbf{a}_1 (y_i - \mathbf{m}_1))$$

should be maximized at each stage of the algorithm. The estimation for  $\alpha_{1L}$  and  $\alpha_{0R}$  are the values of the estimates for  $\alpha_1$  and  $\alpha_0$  at the last updating stage.

The EM algorithm includes the following steps:

### Initialization:

For each  $y_i$  denote the gray-level of the original pixel by  $g_i$ . Using the XOR modulation scheme the halftone pixel corresponding to  $g$  has probability  $g$  to be 0 and  $(1 - g_i)$  to be 1. We define  $p^0(0|y_i) = g_i$  for each  $i \in I_M$ .

### Updating step:

$$E_0(j) = \frac{\sum_{i \in I_M} p^j(0 \mid y_i)(y_i - \mathbf{m}_0)}{\sum_{i \in I_M} p^j(0 \mid y_i)} \quad \text{and} \quad E_1(j) = \frac{\sum_{i \in I_M} p^j(1 \mid y_i)(\mathbf{m}_1 - y_i)}{\sum_{i \in I_M} p^j(1 \mid y_i)}$$

where  $p^j(1|y_i) = 1 - p^j(0|y_i)$ .

Set  $\alpha_0(j) = 1/E_0(j)$  and  $\alpha_1(j) = 1/E_1(j)$  then:

$$p^{j+1}(0 \mid y_i) = \frac{p^0(0 \mid y_i) \mathbf{a}_0(j) \exp\{-\mathbf{a}_0(j) y_i\}}{p^0(0 \mid y_i) \mathbf{a}_0(j) \exp\{-\mathbf{a}_0(j) y_i\} + p^0(1 \mid y_i) \mathbf{a}_1(j) \exp\{-\mathbf{a}_1(j) y_i\}}$$

## 6.2 Scoring code words

Given the probability of each pixel in the scanned image  $S$  to originate from a black (white) pixel, the appropriate code words can be scored according to the Maximum Likelihood (ML) principle.

Let  $w=(w_1, \dots, w_L)$  be a code-word form the codebook of a given sub-image  $p=(p_1, \dots, p_L)$  in  $I$ . Assume that the corresponding scanned sub-image in  $S$  has discriminative function values  $y=(y_1, \dots, y_L)$ . The ML score for  $w$  is given by  $s(w)=P(w|y)$ . Since each code word has equal a-priory probability this score is equivalent, under the assumption that observations are stochastically independent, to  $s(w) = P(y | w) = \prod_{i=1,L} P(y_i | w_i)$ .

The log likelihood score is given by:

$$scr(w) = \log P(y | w) = \sum_{i=1,L} \log P(y_i | w_i) \cong \sum_{i=1,L} (-1)^{w_i} \log( P(y_i | 0) / P(y_i | 1) ) \quad (2.5)$$

where  $\cong$  means - equal up to a constant that does not depend on  $w$ .

Substituting  $P(y_i|0)$  and  $P(y_i|1)$  in (2.5) for the expressions of the bi-modal non-symmetric exponential distribution estimated before the score for  $w$  becomes

$$scr(w) = \sum_{i=1,L} (-1)^{w_i} F(y_i) \text{ where}$$

$$F(y_i) = \begin{cases} \log \left( \frac{\mathbf{a}_{0R}}{\mathbf{a}_{1R}} \right) + \mathbf{a}_{1R}(y - \mathbf{m}_1) - \mathbf{a}_{0R}(y - \mathbf{m}_0) & y > \mathbf{m}_1 \\ \log \left( \frac{\mathbf{a}_{0R}}{\mathbf{a}_{1L}} \right) + \mathbf{a}_{1L}(y - \mathbf{m}_1) - \mathbf{a}_{0R}(y - \mathbf{m}_0) & \mathbf{m}_0 < y < \mathbf{m}_1 \\ \log \left( \frac{\mathbf{a}_{0L}}{\mathbf{a}_{1L}} \right) + \mathbf{a}_{1L}(y - \mathbf{m}_1) - \mathbf{a}_{0L}(y - \mathbf{m}_0) & y < \mathbf{m}_0 \end{cases}$$

The code word that achieves the highest score  $scr(w)$  is chosen for decoding the message  $M$ .

## Appendix A – Theoretical analysis of Visually Significant 2-D Barcode

We consider a coding scheme that produces a 2-D barcode patterns with visual resemblance to a pre-specified 2-D gray scale image. This scheme is termed Visually Significant Barcode (VSB).

### A.1. VSB coding scheme:

Let  $I$  be an  $(m \times l)$  gray level image, and let  $M$  be a (binary) message to be transmitted.

For the sake of simplicity we assume that  $I$  can be divided to sub-images of fixed size –  $(n \times n)$ . By mapping each sub-image to a binary  $(n \times n)$  array a halftoned version, denoted  $B$ , of the original image is produced. The message  $M$  can be encoded in  $B$  in the following way:

Given a collection of distinct halftone mappings the message  $M$  determines uniquely, the half-tone mapping for each sub-image. Hence, each message produces a distinct

half-toned image  $B(M)$ . For decoding, one needs to identify the mappings used for half-toning and then one can reconstruct the message. This procedure is possible since the decoder has the original image and therefore can extract the half-tone mappings by comparing it to the half-toned version produced uniquely by the message.

In order to achieve visual resemblance between the original image and its half-toned version, the half-tone mappings should preserve the visual content of the gray scale sub-images. At the same time, in order to allow unique decoding these mappings should be distinct. Moreover, since the decoder receives the half-toned image with some quality degradations, the half-tone mapping should not only be distinguishable but also distant enough (with respect to a suitable measure) to allow robust decoding procedure.

The preceding discussion implies that the VSB coding scheme has an intrinsic trade-off between visual fidelity and reliable transmission rate. We present a mathematical description of the VSB coding scheme that formulates this trade-off.

### **VSB mathematical model**

Let  $p = (p_1, \dots, p_L)$ , where  $L = n^2$ , be a gray scale sub-image, that is an  $(n \times n)$  pixel array. The corresponding set of output binary arrays,  $C(p) = \{w^1, \dots, w^N\}$ , where  $N = |C(p)|$ , constitutes a code-book (a set of code words. Note that this set depends on the gray-level pattern of the sub-image  $p$  since the visual content is to be approximated by all binary code-words. For the rest of this section we fix a sub-image and consider the properties of the corresponding code-book in relation to visual requirements and transmission rate considerations.

#### **Definition: Capacity**

The capacity of the code is defined as  $\text{Cap}(C(p)) = \log_2(|C(p)|)/L$  bits per symbol.

#### **Features of the VSB code book:**

(1) The members of  $C(p)$  should be well separated with respect to some distance measure pertaining to the noise contaminating the transmitted half-toned sub-image, e.g. Hamming distance. We denote this distance measure by  $D_c$  and require that for some constant  $d$ :

$$D_c(w^i, w^j) \geq d, \text{ for each } i \neq j, i, j = 1, \dots, N.$$

In order to preserve the visual content of the gray-scale sub-image, we impose two sets of constraints on the code-book:

(2) The average gray level of each code word in  $C(p)$  should be as close as possible to the average gray level of  $p$ :

$$\frac{1}{L} \left| \sum_j w_j^i - \sum_j p_j \right| < K \text{ for each } i, \text{ where } K \text{ is a constant.}$$

(3) The average (over all code words) gray level of each binary pixel should equal the gray level of the corresponding pixel in  $p$ . Under the assumption that code words are drawn uniformly this constraint implies that:

$$\frac{1}{N} \sum_i w_j^i = p_j \quad \text{for each } j=1,\dots,L.$$

Under constraints (1), (2) and (3), a good codebook is one that minimizes a distance measure that implies good visual quality.

In the sequel, we propose three approaches for finding good codebooks, but prior to this we give a rough estimation for the VSB capacity:

### Estimating the VSB capacity

We consider, as a starting point, sub-images of constant gray level  $g$  and assume that  $k=g*L$  is an integer. In this case constraint (2) is satisfied with  $K=0$  if there are exactly  $k$  ones in any of the code words. Such codes are termed - constant weight codes. Restricting the discussion to the case  $d=2$  in constraint (1) the capacity of the code is:

$$cap(C(p)) = \frac{\log_2 \binom{L}{k}}{L}$$

Substituting  $k=g*L$  and using Stirling approximation for  $n!$  one gets:

$$cap(C(b)) \cong H(g) = g * \log_2(g) + (1-g) * \log_2(1-g), \quad \text{where } \cong \text{ means asymptotically equal for large } L.$$

It turns out (to be proven rigorously) that  $cap(C(p))$  is an increasing monotone function of  $L$  and an increasing monotone function of  $g \in (0,0.5]$ . The optimum capacity, for each  $L$  is achieved at  $g=0.5$  where it is equal to 1. However, if all sub-images in an image have constant gray level, then this image has no visual significant. Therefore, at least two gray levels should be allowed and the capacity is strictly smaller than 1. Choosing the gray levels  $g$  and  $1-g$  the capacity of the code is asymptotically equal to  $H(g)=H(1-g)$ .

The gray levels  $g$  and  $1-g$  should be separated enough to allow visual distinction. On the other end  $g$  should be close to 0.5 to increase capacity. A reasonable compromise is to choose gray level values in the interval (0.25-0.35). We comment that for  $g$  in (0.25-0.35) the  $H(g)$  bound is reached to within 10% for  $L$  values of 16 (4x4 array) to 25 (5x5 array).

Remark: We have seen that the capacity is  $H(g)$  only in those cases that each sub-image has constant gray-level ( $g$  or  $1-g$ ). We will show in the sequel that the same capacity can be achieved for mixed ( $g$  and  $1-g$ ) sub-images.

### Constructing VSB codebooks.



We return now to the problem of finding a good VSB codebook. Unfortunately, this problem is rather difficult, due to the big number of constraints implied by constraint (1). In what follows we propose three heuristic approaches for constructing such codebooks.

### **Sub-Optimized VSB codebook.**

A VSB codebook can be constructed by a greedy algorithm that adds one code word at each stage. Since an iterative algorithm can't satisfy all the constraints imposed on the codebook at each stage, these constraints are replaced by related objective functions. The new code word is chosen to be the optimal word with respect to a visual objective function (pertaining to constraints (2) and (3)) that satisfies constraint (1) in relation to all the code words previously chosen.

This approach suffers from two main drawbacks:

- (a) It does not guarantee finding an optimal or even a sub-optimal codebook.
- (b) Encoding and decoding with different codebooks for each gray-scale sub-image is hardly practical, except for very small values of L.

Due to these drawbacks we have not pursued this approach.

### **Computable codebooks.**

In this approach, the code word is computed on the fly, for each sub-image, using a suitable pre-defined halftone function that maps gray-scale sub-images to binary arrays. This function depends on the message M (as a parameter) and produces unique halftone patterns for different messages. We have considered two types of half-tone functions:

**Dithering matrices** – arrays of gray-scale values that act as thresholds on the input sub-image. Given a collection of such matrices, a matrix can be chosen, for each sub-image based on the message to be encoded. This approach can handle multi-gray level images. Unfortunately, it is possible to prove that the number of dithering matrices that produce unique half-tone patterns even for relatively small sub-images is restricted to below the VSB capacity computed in section 1.2.

**XOR patterns** – binary arrays that produce half-tone patterns by XOR operation. This approach is limited to images having only 2 gray levels: g and 1-g. Each sub-image is represented by an (n x n) array of ones and zeros, where zero represents the low input gray level and one the high input gray level. Assuming that  $k=g*L$  is an integer, the “generating” codebook consists of (n x n) binary arrays with k ones and L-k zeros. For each sub-image - p, a generating code pattern - v is chosen (by the message M) and the code word w is the XOR of these two arrays  $w = b \oplus v$ .

#### Proposition:

The VSB generated by XOR patterns code-book that consists of all (n x n) binary arrays with exactly k ones, has the following properties:

- (a) The hamming distance between each couple of code words is greater or equal to 2.  
(This fact relates to constraint (1) with  $d=2$ ).
- (b) Constraint (2) is satisfied with  $K = 2 * (g - g^2)$ .
- (c) Constraint (3) is satisfied.

(d) The capacity of the XOR code is equal to  $\frac{\log_2 \binom{L}{k}}{L}$

Proof:

(b) Since the number of “ones” in each XOR pattern is exactly  $k$ , the distance between any different two patterns is greater or equal two. It is easy to verify that the XOR operation preserves this property.

(c) The worst value for  $\left| \sum_j w_j^i - \sum_j p_j \right|$  is reached when the sub-image has  $L-k$  zeros and all the  $k$  ones of the Xor pattern correspond to sub-image ones. In this case the output pattern is all zeros, but the average gray level is:  

$$\frac{1}{L} \sum_j p_j = \frac{1}{L} g * (L-k) + (1-g) * k = 2 * (g - g^2)$$

(d) The set  $\{v_j^i\}_{i=1, \dots, N}$  of entries in location  $j$  of all Xor-patterns has ones to zeros ratio of  $k$  to  $n-k$ . Since xoring a bit with zero preserves its values and xoring it with one flips its value it follows that:

$$\text{For a zero bit in location } j: \frac{1}{N} \sum_i w_j^i = \frac{1}{N} \sum_i v_j^i \oplus 0 = \frac{c * k}{c * L} = g ,$$

$$\text{and for a one bit in location } j: \frac{1}{N} \sum_i w_j^i = \frac{1}{N} \sum_i v_j^i \oplus 1 = \frac{c * (n-k)}{c * L} = 1 - g \text{ for each } j,$$

$$\text{where } c \text{ is the ratio } c = \frac{N}{L} .$$

(e) See section 1.2 €

The XOR patterns approach has the advantage of simplicity and high capacity. The corresponding VSB capacity, reaches asymptotically the bound of  $H(g)$ . However, the visual quality might suffer from XOR patterns that have blobs of ones or zeros. Hence it is preferable that such unpleasant patterns will be removed from the codebook. This removal reduces the capacity of the code, but improved the visual fidelity.

### **Tiling with small binary arrays (constrained system).**

This approach can be utilized as a method for half-toning multi gray-level images or for improving visual quality of the XOR patterns approach. We demonstrate it by considering tiling with  $(2 \times 2)$  binary arrays and limit the discussion to images with only two gray-levels: 0.25 and 0.75. Generalizations to higher dimensions and multi gray-level images are conceivable.

Given an image with gray levels in the set  $\{0.25, 0.75\}$ , each pixel is mapped to a  $(2 \times 2)$  binary array, in such a way that the average pixel intensity is preserved. To improve

visual fidelity, some constraints are imposed on the way arrays are tiled together. These constraints are designed to prevent the production of unpleasant patterns such as blobs of ones or zeros. The tiling method with  $(2 \times 2)$  arrays can be viewed as an algorithm to remove unpleasant patterns from a codebook with higher dimensionality.

The tiling algorithm can be described as a finite state system defined on a directed graph whose vertices are the tiling arrays and whose edges represent the permitted transitions. Encoding is implemented by selecting, at each tiling step, one of the outgoing edges of the current vertex, according to the input message. At decoding, the edges are reconstructed from the original image and its encoded version.

The performance (capacity) of the Tiling approach to VSB can be analyzed via the theory of constrained systems. Under suitable conditions, the capacity of a one-dimensional constrained system can be computed from the biggest eigenvalue of the adjacency matrix of the graph describing the system. The proposed tiling system is two-dimensional; hence our analysis provides lower and upper bound for the capacity.

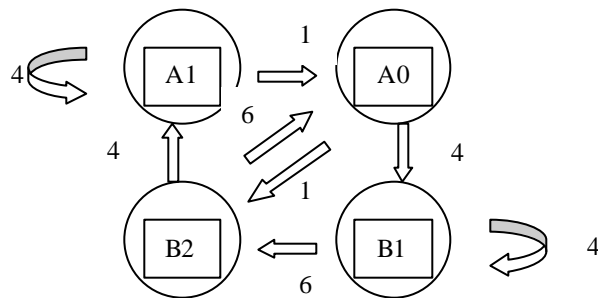
To demonstrate this analysis we construct an example of a constraint code graph for the case of constant gray level (0.25) image and compare its capacity to the estimation presented in section (1.2). An equivalent graph can implement the code for 0.75 gray level images. Appropriate switching between these two graphs implements the code for the two gray-level images.

(2 x 2) Tiling example

To achieve the average gray level of 0.25 we use the following binary arrays:

- 1 array with no 1's – represented by vertex A0 ,
- 4 arrays with one 1 – grouped in each of the vertices A1 and B1,
- 6 arrays with two 1's – grouped in vertex B2.

The following diagram describes the system graph:



The digit on each edge indicates the number of outgoing edges from each vertex inside the group.

The adjacency matrix is:  $\begin{pmatrix} 0 & 0 & 4 & 6 \\ 1 & 4 & 0 & 0 \\ 0 & 0 & 4 & 6 \\ 1 & 4 & 0 & 0 \end{pmatrix}$  and its biggest eigenvalue is 6.4495.

Hence the capacity of the corresponding one dimensional tiling code is  $\log_2(6.45)/4 = 0.672$ ; This value is an upper bound for the capacity of the tiling system, since the one dimensional system is less constrained than the two dimensional one. The difference being that the one-dimensional code is not constrained by vertical tiling limitations.

Computing a lower bound can be achieved by considering a two dimensional tiling system that uses the one dimensional system for odd rows and tiling with A1 arrays only for even rows. It is easy to verify that this system satisfies the constraints imposed on the original two dimensional system but has lower capacity which is the average of the two one dimensional tiling methods. This average is given by:  $\frac{1}{2}(0.67 + 0.5) = 0.585$ .

To conclude, the capacity of the constrained tiling system satisfies  $0.585 < \text{cap} < 0.672$ . This result can be compared with the 0.5 capacity of the (2 x 2) XOR code and the 0.677 capacity of the (4 x 4) XOR code, both corresponding to the same gray level of 0.25.

Remark:

In order to trade capacity with visual quality, one can remove certain edges from the code graph, e.g. those that tiles two arrays with adjacent ones.

#### 4. Reference

- [1] Carl Staelin, and Larry McVoy, "mhz: Anatomy of a micro-benchmark", in Proceedings of USENIX 1998 Annual Technical Conference, pp. 155, New Orleans, Louisiana, June 1998.