Dynamic Optical Braille Recognition (OBR) System
A. marwa.abdelmonem.ali@gmail.com, B. mohammad.hamdy@yahoo.com, D. kemara@eun.eg and E. habiba.elgebaly@gmail.com

1Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt

Abstract-Braille is one of the most important means of written communications between visually-impaired and sighted people, so it gains the research interest. This paper describes a new technique for recognizing Braille characters in Arabic double sided Braille document. The main challenge resolved here is to build up a complete OBR system that is completely invariant to scale of the scanned image starting from scanner passing image enhancement stages and followed by stages to detect parts of dots, then detecting the whole dot, and finally the Braille cell recognition. This technique can be applied regardless the grade of the Braille document (Grade one or Grade two). Besides, the proposed stages up to the Braille cell recognition can be used in recognizing Braille documents written in other languages too.

Keywords: Image processing, Optical Braille Recognition OBR, Braille

1. Introduction
Communication in the written form plays a very important role in the daily life for many people like in the educational purposes or taking notes etc. However, visually impaired or blind people face problems to deal with these written documents. Braille is the most commonly used writing convention for visually impaired people.

Using new technology, producing Braille documents is somehow easy; the remaining problem is how to digitize and recognize Braille documents. This problem is worthy to solve for two reasons; first, there is a wealth of books and documents that only exist in Braille which need be digitized and preserved. Second, there is an everyday need for duplicating (the equivalent of photocopying) Braille documents and translating them into natural written language.

Braille characters (cells) consist of six dots arranged in the shape of rectangle with two columns and three rows as shown in Fig.1. This generates 64 combinations that can contain the alphabet and punctuation marks.

Dot height is approximately 0.02 inches (0.5 mm); the horizontal and vertical spacing between dot centers within a Braille cell is approximately 0.1 inches (2.5 mm); the blank space between dots on adjacent cells is approximately 0.15 inches (3.75 mm) horizontally and 0.2 inches (5.0 mm) vertically. Braille documents can be written in two different ways: grade 1 and grade 2. In grade-1, there is a one to one correspondence to the ordinary characters, but in grade-2 there are conventions for representing whole words of printed characters by a single Braille character.

The approach described here shows the feasibility of a cost-effective and easy to use Braille reading system. It does not require expensive custom made and complicated hardware [3]

This paper is organized as follows. In this section 2, a brief explanation of the OBR system used in related work. In section 3, the contribution of this work is presented. In section 4, system results are shown. Section 5 presents conclusion and future work..

2. Related work
In this section a brief explanation of related work [1] is presented. Figure 2 shows a block diagram for system processes.
OBR systems begin with a simple scanning process using an ordinary flatbed scanner. Then, a noise removal process is performed on the scanned image to remove sharpness and illumination problems that appear from scanning process. Other preprocessing steps are used as shown in the next section.

2.1 Pre-processing

1. Converting the image to gray level [1]

![Fig. 3. Image converted to gray](image)

2. Image thresholding [1]

![Fig. 4. After threshold](image)

2.2 Dot parts detection

After dividing the image into three colors (dark, bright, gray) as in figure 4, the next step is to determine the recto and verso dots.

Authors of [1] use static number of pixels to detect the dots in the image.

As most of Braille images - in common resolution- the dot height is 8 pixels; each dot is composed of a bright and a dark region with a small space between them. The implication is that if the bright region comes at top where the dark one comes at bottom then this is a recto dot, the contrary situation results in a verso dot.

![Fig. 5. Recto (Gary dots) and Verso (White dots)](image)

2.3 Whole parts detection

This step is for removing the noise and overlapping resulted from dot parts detection phase and for separating recto and verso dots into two different buffers to easily recognize the cells [1].

![Fig. 6. The image contains recto dots only](image)

2.4 Braille cells recognition

This step does not depend on fixed lengths for cells heights and distances between them.

The main idea in this step is to determine the number of rows and columns [1]

![Fig. 7. Recto and verso dots in two different colors](image)

2.5 Conversion to text

Each Braille character is coded into a six bit binary number based on the presence or absence of dots. The presence of a dot is indicated by a 1 and its absence by a 0.

Simply this step is a simple one to one mapping from a database containing the binary code and the letter corresponding to this representation

![Fig. 8. Conversion to Binary](image)

![Fig. 9. Conversion to text](image)

3. Dynamic OBR system

The Contribution here serves as modification in system represented in [1] to be completely dynamic. The contribution was in two ways. Either by enhancing phases in related work or adding a new phase as next explained

3.1 Pre-processing

1. After scanning Brialle document, histogram equalization is applied to enhance the quality of the image. Furthermore, it is an effective mechanism to make the image consistent with thresholding process proposed in[1]

2. After thresholding is applied, a new step is added which is applying flood fill algorithm with 4-neighbour [5], [2]. This step is used later in next phases to make them dynamic. The output of interest of flood fill is calculation
of the average dot height and width to make the algorithm invariant to different dot sizes.

\[
\text{AvgDotW} = \text{Floor}(\text{average width of Flood fill regions})
\]

\[
\text{AvgDotH} = \text{Floor}(a * (2 * \text{average height of Flood fill regions}))
\]

Where most appropriate value of \(a\) is 0.969696 is the constant estimating actual dot height relative to flood fill regions that represent about half of the dot.

### 3.2 Dot parts detection

Searching for dot parts is performed using vertical window of size equals to \(\text{avgDotH}\) on the each column in the image starting from top to bottom. This is done such that:
- If pixel \((1) + \text{pixel (2)} > 0\) AND the last two pixels in the window < 0 then this column is a part of a recto dot.
- If pixel \((1) + \text{pixel (2)} < 0\) AND last two pixels in the window > 0 then this column is part of a verso dot.

In case of having any of the two conditions true the dot part is marked in the image and move the start of the window down with \(\text{AvgDotH}\)*12/8 since this is the vertical space between two dots. Parts of rector dots are marked with different color value than parts of verso dots. Eventually, we have Figure 10 as visualized output of dot part detection.

![Output of dot part detection](image1)

Fig. 10. Output of dot part detection

Having done this, the result will be recto and verso dots with some noise and overlapping areas so the next phase (i.e. the whole dot detection) is considered to remove them and So two different buffers are generated one contains recto dots and the other contains verso dots. The verso one is flipped horizontally while the recto is not changed. Then each of them passed to next phases to recognize Braille character on each size.

### 3.3 Whole dot detection

Having applied these steps in the same manner presented in [1], Problem of duplicating dot or missing dot appears. However changing the threshold somewhat improves the result, it still fails in many other cases. For example, it fails to be generalized as a scale of the mechanism presented in [1].

To avoid removing actual dots, or redundancy of dots, the average dot height is used as follows:

- A vertical search is performed on the image from up to down. Then, check if the pixel color value (i.e. white for recto, gray for verso) remains the same for \((5 * \text{AvgDotH/8})\) pixels. If check succeeds, such vertical area is considered as one dot. Doing this vertical search on all columns of the image results in identifying the count of dots in each column with 1 pixel width.

- As a result, a signal is obtained containing number of dots in each column along the y axis and column index along the x axis. Plotting such signal shows these values increase and decrease as illustrated in figure 12.

- As shown in the figure peeks indicate correct columns. Therefore detection of a valid column can be determined by taking peeks within window of average dot width (\(\text{AvgDotW}\)). After a valid column has been detected, each dot belongs to this column is stored as 3*2 pixels drawn in the new image.

![Increasing decreasing curve](image2)

Fig. 12. Increasing decreasing curve
(This numbering under the curve indicate the number of dots in each column in the image)

After this step the resultant image is containing 3*2 points which represents the dots in the correct position. Figure 13 shows the result with no duplication, missing dots or extra dots (noise).

![Whole dot detection output](image3)

Fig. 13. Whole dot detection output

### 3.4 Braille cells recognition

Vertical and horizontal projections are performed [1] as illustrated in figure 14, the goal of such projection is to partition the document into Braille characters such that each character consisting of two dots width and three dots height (see figure 15). This can be achieved by segmenting vertical projection into groups of 3 dots and segmenting horizontal projection into groups of 2 dots. Then, each horizontal group is merged with each vertical group forming cells exist in the document. But what if there is a complete column of missing dots. For example, as seen in figure 16, there are missing columns where the simple method presented above can be
easily violated. The following is a proposed technique that helps solve this problem.

Fig. 14. Vertical and horizontal projection

Fig. 15. Character to be converted

Fig. 16. Missing character

In columns, average dot width can be used to dynamically estimate the distances between columns by using the following equations:

\[
\begin{align*}
\text{wcd} &= \text{ratio}_{\text{wcd}} \times \text{AvgDotW} \\
\text{icd} &= \text{ratio}_{\text{icd}} \times \text{AvgDotW}
\end{align*}
\]

Where wcd (Within-Character-Distance) is an approximate distance between two columns forming the character cell, icd (Inter-Character-Distance) is an approximate distance between end of second column of the character and the beginning of the first column of following character.

Most appropriate values based on analysis on the document ratio_{icd} = 1.59 & ratio_{wcd} = 0.6. This is because these distances are linearly proportional with width of Braille dot.

There are three cases that need to be considered:

- The first column in the character is missing
- The second column in the character is missing
- The two columns are missing as shown in figure 18. This character is considered as a space between two words.

A horizontal search is performed on the horizontal projection starting from left to right. Assuming the first column of the first character exists. This assumption results in two cases. The first case is when the two columns exist (figure 17), move right with the actual distance. This distance is equal to the space between these two columns. The second case when the second column is missing (figure 17) move distance equal to wcd + (2\*AvgDotW) along with save the width of this character. This equation (i.e. wcd +2\*AvgDotW) is the estimated cell width that can be used also to estimate a missing character. The icd can be used to check the missing column of next character if the distance is within icd ± AvgDotW/2.

Regarding the rows, we count three dots then this distance is considered as the character height. It’s rare to find a complete row with no dots so problem of missing row rarely exists.

At the end, all character widths and heights will be calculated and saved, and then we can easily separate characters as in figure 18.

Fig. 17. Sample vertical projection output

Fig. 18. Cell recognition

4. Experimental results

The Dynamic OBR system has been tested against a dataset of 10 Braille documents. Document size is about 200 characters. The data is scanned using a normal flat scanner. Then, these scanned images are processed through system phases. After each phase, we calculate its accuracy by comparing its result with the ground truth passed from the previous phase. We depend on number of correctly detected dots as our accuracy measure.

The following table shows the accuracy of each phase

<table>
<thead>
<tr>
<th>Phase</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold phase</td>
<td>99.5%</td>
</tr>
<tr>
<td>Dot parts detection</td>
<td>98%</td>
</tr>
<tr>
<td>Whole part detection</td>
<td>97.8%</td>
</tr>
<tr>
<td>Braille cell recognition</td>
<td>98.5%</td>
</tr>
<tr>
<td>Conversion to text</td>
<td>99.7%</td>
</tr>
</tbody>
</table>

5. Conclusion and Future work

The proposed system has used new techniques to recognize Braille cells using a standard flat scanner. As a whole, the approach described here shows the feasibility of a cost-effective Braille reading system. It does not require expensive custom made and complicated hardware. It uses a flat-bed scanner which can be shared with other applications, inter-point Braille documents are handled in addition to single sided ones.
In the future, we aim to implement more features. The system should be able to interpret more than one language. Also, the system should be able to translate any Braille document regardless its grade, such as grade 2. On the other hand, the system performance could be improved to be integrated with text to speech engine and support real time processing.

6. References