International Journal on Advanced Science Engineering Information Technology

Vol.7 (2017) No. 4-2
ISSN: 2088-5334

Braille2Alpha: Braille Dots Recognition with Alphanumeric Conversion

Abidah Zainal[#], Aida Mustapha[#], Mohd Zainuri Saringat[#], Pravind Raja[#]

[#]*Soft Computing and Data Mining Centre, Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Parit Raja, 86400 Batu Pahat, Johor, Malaysia*
Email: aidam@uthm.edu.my

***Abstract*— The Braille system is a tactile language where each element is represented by a cell with six dot positions, arranged in three rows and two columns. Each dot position can be raised or otherwise allowing different configurations which can be felt by trained fingers. For caregivers with normal sights, training their fingers is not naturally easy. This paper presents an image processing approach to Braille dots recognition with alphanumeric conversion, whereby a person with normal eyesight is able to translate a raw Braille image captured by a mobile camera into the corresponding alphanumeric texts. Image pre-processing relies heavily on various segmentation algorithms in the MATLAB image processing Toolbox. This application is hoped to help the blind and the visually impaired community as well as their caregivers to translate Braille texts digitally, hence enhancing the communication and collaboration possibilities.**

***Keywords*— image processing; Canny algorithm; Braille; Android**

I. INTRODUCTION

Braille is a system of touch-reading and writing for the blind or those who are visually impaired. The system consists of raised dots that represent the letters of each of the alphabet. Even though most people are blessed with normal or good eyesight, people tend to take their ability to see for granted by ignoring the little details in life that truly make a difference. In reality, blind or visually impaired children need to learn how to read and write using the Braille alphabets from a very early age. However, for some children, the Braille text might not be an immediate choice because learning Braille takes a long time without the help of the experts.

The practice of dot-touched communication was first introduced by Charles Barbier who served in Napoleon Bonaparte's French army in the early 1800s [1]. He was the creator of 'night writing' system or the sonography.The Braille system is a tactile language where each element is represented by a cell with six dot positions, arranged in three rows and two columns. Each dot position can be raised or otherwise allowing different configurations which can be felt by trained fingers [2], [3]. Fig. 1 shows the Braille dots coordination and the printed Braille's dots.

The Braille system had been used widely, and the mappings or sets of character designations vary from language to language. By using the Braille alphabet, blind people or those with visual-impaired as well as normal persons can review and study the written word. In English Braille there are three levels; (1) Grade 1 – a letter-by-letter transcription used for basic literacy, (2) Grade 2 – an addition of abbreviations and contractions, and (3) Grade 3 – various non-standardized personal shorthand [4].

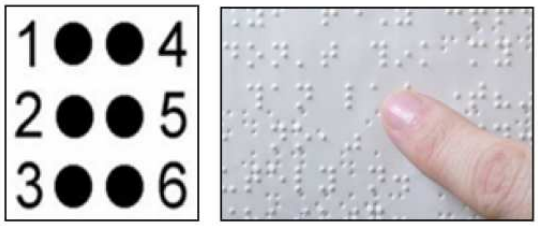

Fig. 1 Braille cell and the embossed Braille alphabet

The main challenge within the blind and the visually impaired community is mastering the Braille text, so they are able to understand and access information and knowledge in their early years. However, these configurations need to be memorized, so it is very difficult to learn especially among the caregivers. The problem arises because the majority of the community is unable to understand this medium of communication.

To address this issue, this project is set to design and develop an application that is able to convert the image of Braille texts directly into words. With this application, users

only need to capture the image of Braille texts to get the result of the corresponding meaning. This application is also able to teach people with normal vision to understand Braille texts so that information can be shared and access freely. However, note that this work will only focus on the object translation of 'Uncontracted Braille' or Grade 1 English Braille and is targeted for the use of novice Braille users, The Grade 1 English Braille characters is shown in Fig. 2.

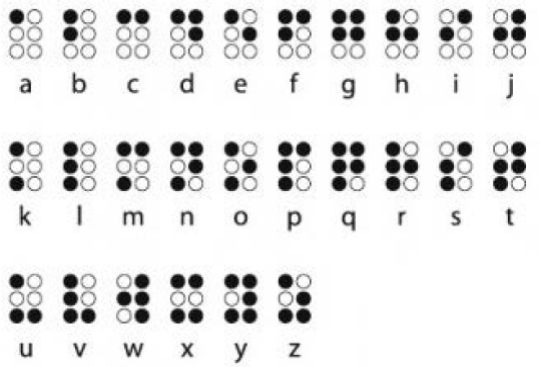

Fig. 2 Grade 1 English Braille characters

The Braille2Alpha application is intended to automatically convert an image of Braille text into its corresponding alphanumeric text without the expert knowledge of the Braille system. The most basic yet important feature is the capability of detecting the Braille input captured from a standard mobile phone camera. This is plausible through the edge detection algorithm in image processing.

Image processing concerns on operations that manipulate a digitized image, which can be in the form of a single image, a series of images extracted from a video or a single video frame. A digitized image is composed of a finite number of elements, each of which has a particular location and value and these elements are referred to as picture elements, image elements, pels, and pixels. Pixel is the term most widely used to denote the elements of a digital image [4]. An image processing system pre-treating images as two-dimensional signals while applying a set processing method such as grayscale conversion, binary conversion, filtering, edge detection, circle detection, image projection and morphological dilation.

Edge detection algorithm is important to significantly reduce the amount of data in an image while preserving the structural properties to be used for further image processing. In this project, three edge detection algorithms will be reviewed, which are the Canny [5], Sobel [6] and Prewitt [7] algorithms. The work of Canny algorithm starts by detecting the probability of real edge points that should be maximized as well as the probability of falsely detected non-edge points. This means the detected edges should be as close as possible to the real edges and one real edge should not result in more than one detected edge. This essentially corresponds to maximizing the signal-to-noise ratio.

The Sobel operator is also one of the methods used in edge detection of image processing. The Sobel operator works by performing a 2-D spatial gradient measurement on images. This means the operator finds the approximate magnitude of the absolute gradient at each point from the input image in greyscale. Then, the Sobel edge uses a pair of 3x3 convolution masks to perform two things. The first is to estimate the gradient in the *x*-direction (columns) while the second is to estimate the gradient in the *y*-direction (rows). Because the convolution mask is much smaller than the actual image, the mask is slid over the image, focusing on a block of pixels at a time [8].

The Prewitt operator is also used in the edge detection of image processing whereby it is able to detect two types of edges, i.e., horizontal and vertical edges. Edges are calculated by using the difference between corresponding pixel intensities of an image. All the masks that are used for edge detection are also known as derivative masks.

Because an image can be treated as a signal, therefore changes in a signal may be calculated using differentiation. This is the reason the Prewitt operator is also known as the derivative operators or derivative masks. Table 1 shows the comparison between these algorithms and which ones are the best algorithms for processing a noisy image and yielding the best results.

TABLE I
COMPARISONS BETWEEN EDGE DETECTION ALGORITHMS

	Sobel	**Canny**	**Prewitt**
Method for detecting horizontal and vertical edges	3x3 convolution kernels	Adapts Sobel's 3x3 convolution mask to find the edge strength	Similar to Sobel's operator
Noise detection	Very sensitive to noise thus does not eliminate them completely	Detects, smoothens the image and eliminates noise in its entirety	Equally sensitive to noise just like Sobel and partially eliminates them
Accuracy of image detection	Quite accurate given the fact that it utilizes simplicity in detecting edges and the orientations	Uses localization and improved signal-to-noise ratio hence image is yielded with optimal results	Fairly accurate and produce similar results like Sobel', operator
Probability of finding error rate	Low	High	Low

In determining the best edge detection algorithm for the proposed work, the outputs from the three algorithms were compared based on results generated from the MATLAB software. Fig. 3 shows the comparative results. The Canny algorithm produced a better result as this algorithm takes into account the difference among regions in an image. Canny produced thin lines for its edges by using non-maximal suppression. It smoothens and eliminates the noise thus generating an optimal result.

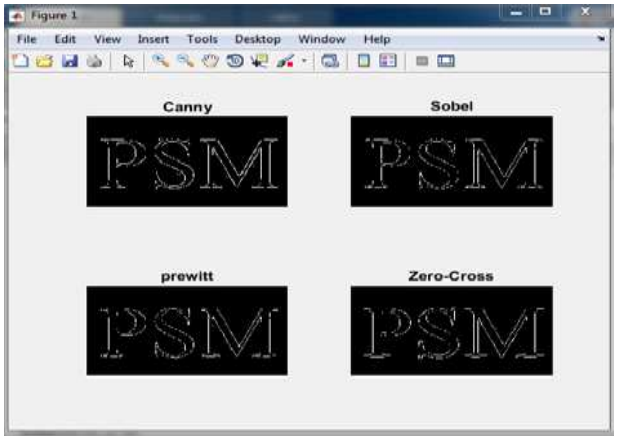

Fig. 3 Outputs from three edge detection algorithms

II. Material and Method

This project adopts the Agile development methodology [9] to integrate the image processing capability on the mobile-based platform. As shown in Table 4, this model involves fundamental processes as in any other process model such as planning, analysis, design, testing, and implementation. However, the model is particularly useful to meet the criteria for collection of innovative, user-centered approaches to system development.

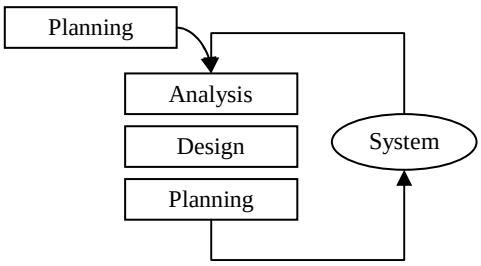

Fig. 4 Agile development methodology [9]

The use case diagram for the Braille2Alpha is shown in Fig. 5, which are capture image, interpret the image and display output.

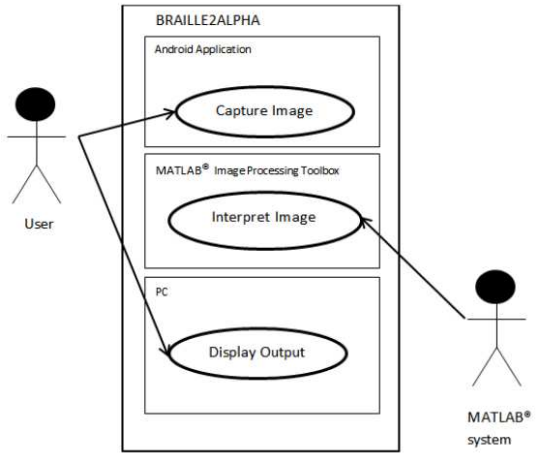

Fig. 5 Use case diagrams for Braille2Alpha

In Fig. 5, the input is in the form of captured images via a phone camera. However, in order to detect the Braille dot within the image, it requires a pre-processing algorithm to convert the RGB-colored image to grayscale, in order to make further processing easier. Edge detection is part of image processing techniques that concerns on detecting the boundaries of objects within images. Braille2Alpha application requires an image processing algorithm with edge detection technique in order to detect the dots in a Braille text so they can be recognized and translated into the corresponding alphanumeric characters. Fig. 6 shows the flowchart for Braille2Alpha application development.

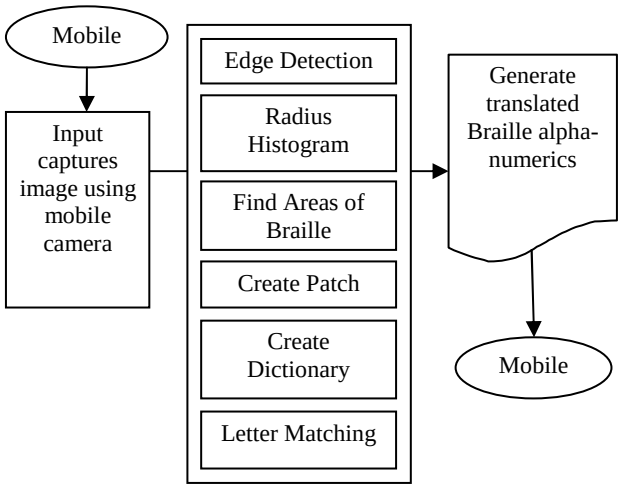

Fig. 6 Process flowchart for Braille2Alpha application

From the figure, the entire application was built on a client-server image processing system. The user captures the input image via the mobile phone camera, which is an Android client. Next, the image is sent to the server via HTTP Internet protocol. On the server, the image is sent to the Android Studio IDE through ASyncTask process and gets converted. Then, A PHP script on the server invokes the server-side application to compute Scale-Invariant Feature Transform (SIFT) in the image [10]. After code computation in the MATLAB software has been completed, the patch image of the Braille alphabet will be translated into alphanumeric alphabet on the mobile device.

The image processing capability was implemented using the MATLAB software with a plug-in library called VLFeat. The input of Braille images was processed using the Canny edge detection algorithm, and the radius was calculated using circle detection, generating an equivalent area of Braille templates to match with the alphabet dictionary. Fig. 7 shows the folder directory that links the sub-functions to the main function of MATLAB® in order to run the script which performs the translation of Braille letter to alphabets.

The inheritance diagram shows the clear picture on how the function script works. In the main Braille.m code, all these sub-functions are called into this single script file via the concept of the data structure. It is also known as passing arguments through function names. This argument will store the function name and the image path. Thus, the code will first access the image path and then call the necessary function needed to translate the Braille script. If the call-back of all functions are done according to the inheritance diagram mentioned earlier, then the execution of the .m code will run without any errors.

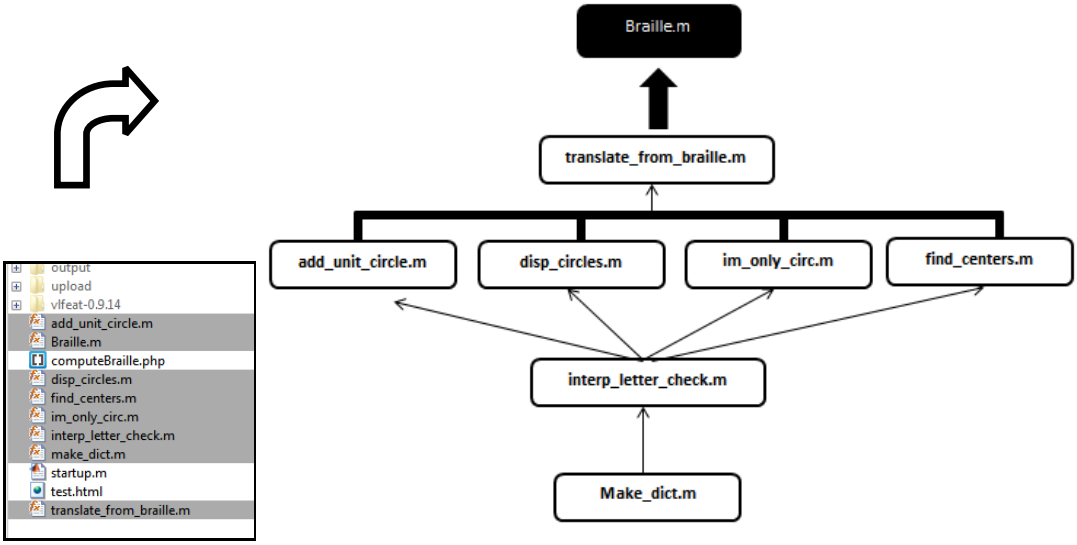

Fig. 7 Inheritance diagram that links the subfunctions to the main MATLAB function

III. Results and Discussion

The Braille2Alpha was developed using Android Studio Integrated Development Environment (IDE), and testing was carried out in each iteration for all modules; capture image, interpret the image, and display output. However, for the sake of brevity, only the final and enhanced iteration's test is discussed.

A. Capture Image

The system begins with a Launcher Activity that will start directly on the startup application logo and proceed to the front page that consists of camera option which given the user a choice to capture the image directly after launch the application. When the user executes the application for the first time, it redirects to camera features as shown in Fig. 8.

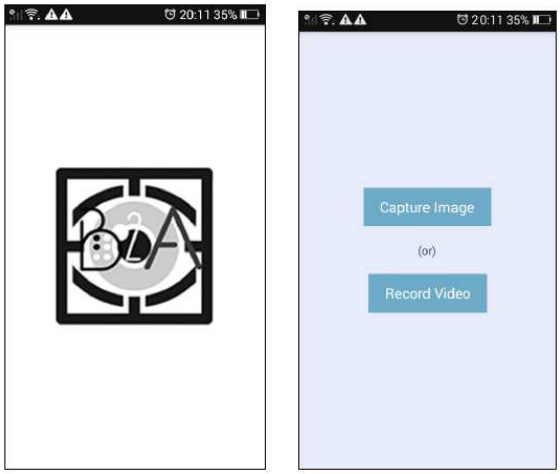

Fig. 8 Capture image interface

When the user clicks on the Capture Image button, the camera feature is activated and the user is allowed to capture the image of Braille to be translated into the next process. The first user must capture the image of Braille script in the distance limit and second flash camera is optional for a better lighting intensity. Fig. 9 shows the camera interface when capturing the Braille image.

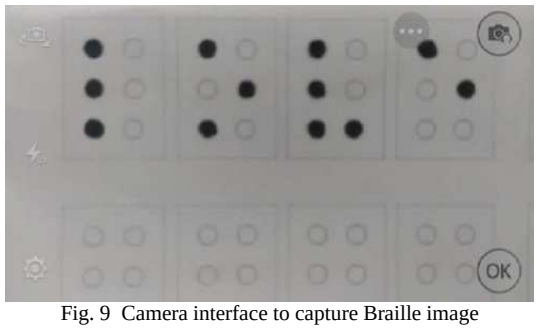

Fig. 9 Camera interface to capture Braille image

Next, the user will be prompted to upload the image to the server for the dots recognition process as shown in Fig. 10. In order for the image to be successfully uploaded, a PHP script called computeBraille.php is implemented inside the system. This PHP script allows a client to upload a captured image by accessing the image path of the previously uploaded image and storing is inside an associative array.

Fig. 10 The upload interface

Table 2 shows the test cases were used to test the Capture Image module. The test results showed that the module passed the tests on capturing the image and uploading to the server.

TABLE II
TEST CASE FOR CAPTURE IMAGE

ID	Requirements	Description	Status
STD_TEST_100_101	SRS_REQ_101	The application shall allow the user to capture the image of braille.	PASS
STD_TEST_100_102	SRS_REQ_102	The image that is being captured must not exceed more than the distance limit between the camera and the Braille script	PASS
STD_TEST_100_103	SRS_REQ_103	The image captured must be under good lighting intensity for a better result	PASS
STD_TEST_100_104	SRS_REQ_104	The application shall allow the user to upload the image to the server.	PASS

B. Interpret Image

The image will then be processed using the MATLAB® codes. The function Braille.m as shown in Fig. 7 works to produce input and output of the image path. Braille.m function is considered as the core process to call other related function as it calls the related parameter codes in MATLAB that calculates the entire algorithms to translate the Braille features in the image. The function for translate_from_braille.m in which it adapts canny algorithm as edge detection and circle detection via Hough Transform calculate and translate the braille image to alphabets or simple letters.

Next, the equivalent letter (alphanumeric text) is matched with the detected Braille cell based on the position and coordinate of the dot placement. A dictionary was built based on the common radius size and patch dimension used. This is important in order to keep the system robust to scale changes. Table 3 shows the test cases were used to test the Interpret Image module. The test results showed that the module passed the tests on initiating the MATLAB® codes, interpreting the image, and using the algorithm to translate the image.

TABLE III
TEST CASE FOR INTERPRET IMAGE

ID	Requirements	Description	Status
STD_TEST_200_101	SRS_REQ_201	MATLAB® system shall be able to startup as soon as the image is successfully uploaded to the server.	PASS
STD_TEST_200_102	SRS_REQ_202	MATLAB® system shall be able to translate the image of braille.	PASS
STD_TEST_200_103	SRS_REQ_203	MATLAB® system shall able to calculate the image using the algorithms implemented in the custom codes.	PASS

C. Display Output

The output of Braille2Alpha application will be displayed on the PC as the main core operation is run in MATLAB® and directly generate the mapping image of alphabet patched on the image of Braille uploaded. The image uploaded took about 21-30 seconds of time elapsed to finally translate and generate the output. Fig. 11 shows the output from the Braille2Alpha application. The output page shows actual input Braille text that has been captured by the camera in given template followed by the result of alphabet patch on top of the image. The accuracy of translating the image depends on the quality of the image taken. If the image is not clear, then the user is prompted to recapture a clearer image.

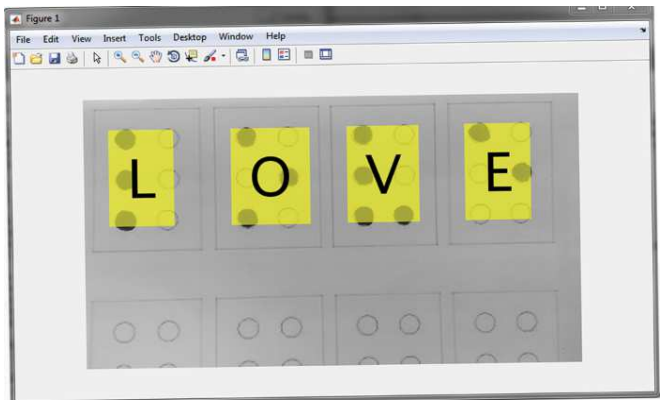

Fig. 11 Output image of Braille script to alphabet

Table 4 shows the test cases were used to test the Display Output module. The test results showed that the module passed the tests on returning the results for user views.

TABLE IV
TEST CASE FOR DISPLAY OUTPUT

ID	Requirements	Description	Status
STD_TEST_300_101	SRS_REQ_301	The system shall allow the user to view the output via MATLAB®	PASS

Overall tests result showed that the system passed all the test cases successfully.

IV. CONCLUSIONS

The Braille2Alpha application comprises of these three core elements such as capture, view, and translates the input Braille image. With the combination of these three elements, this system is capable of helping the users to understand the fundamental of Braille especially the caregivers to translate any Braille script. Based on the testing, the primary benefits of the system are as follows:

- The system provides facilities for visually impaired people and their guardian to translate the Braille in just a few seconds.
- The system is catered to the community to appreciate the contribution of Braille towards individuals with special needs.
- The system is fully automated to translate the Braille image after hitting the translate button.

Nonetheless, Braille2Alpha can be improved in many ways to further enhance its functionality. Some of the identified possibilities for improvement include improving the flexibility of the circle detection algorithm by detecting the dots of Braille without referring to the template provided during testing the prototype. Further, the elapsed time of translating the images as well as the algorithm to detect the actual embossed Braille image can be improved by implementing more specific image integral formula. Finally, the accuracy of the Braille detection capability should be at least maintained even under low light intensity. It is hoped that the application is able to cater the community to appreciate the contribution of Braille towards individuals with special needs.

ACKNOWLEDGMENT

This project is sponsored by Universiti Tun Hussein Onn Malaysia and partially supported by Research Gates IT Solution Sdn. Bhd.

REFERENCES

[1] Mellor, C. M., "Louis Braille: A Touch of Genius", National Braille Press, 2006.

[2] Braille Resources and Information: Braille History. (2016). Retrieved from Braille Works Web site: https://brailleworks.com/braille-resources/history-of-braille/

[3] Louis Braille Biography. (2014). Retrieved from American Foundation for the Blind Web Site: http://braillebug.afb.org/louis_braille_bio.asp

[4] Gonzalez R. C., & Woods, R. E., "Image Processing", Digital Image Processing, 2, 2007.

[5] Canny, J., "A Computational Approach to Edge Detection", IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

[6] Sobel, I., "History and Definition of the Sobel Operator", 2014.

[7] Prewitt, J.M.S., "Object Enhancement and Extraction in Picture Processing and Psychopictorics", Academic Press,1970.

[8] Pitas, I., "Digital Image Processing Algorithms", Hertfordshire: Prentice Hall Europe, 1995.

[9] Kendall. K. E., Kendall, J. E., "System Analysis and Design", Ninth Edition, Harlow: Pearson Education, 2014.

[10] Lowe, D. G., "Distinctive Image Features from Scale-Invariant Keypoints", International Journal of Computer Vision, 60(2): 91-110, 2004.