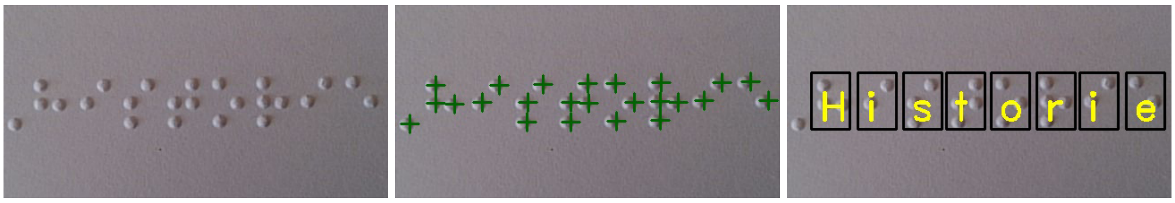
Excel@FIT 2015
http://excel.fit.vutbr.cz

Braille Reader on Mobile Device

Jan Krušina*



Abstract
The aim of this project is to create a portable Braille reader. The main objective is to recognize Braille characters from images taken by camera on a mobile phone, convert them into Latin alphabet, and eventually display the output to the user. Solution of this task is based on a visual detection of Braille characters. Input frames from camera are processed one by one using a special algorithm which separates dots in characters from the rest of the image. Afterwards, dots are grouped into particular characters. Finally, every single character is translated and rendered on screen. This application is capable of detecting dots from books at a very high success rate. Reading from other surfaces, e.g., metal, has a good success rate as well. Thus, the application is able to detect dots on informational signs and other captions, which are commonly used. This reader gives people the ability to read text written in Braille used by blind and visually impaired people all over the world.

Keywords: Braille Font Detection — Braille Reader — Mobile Device

Supplementary Material: Demonstration Video

*xkrusi00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

The main objective is to develop a device that would be able to convert Braille into Latin alphabet. Program that would be possible to use almost everywhere. Program capable of reading complex books and scripts written in Braille or simple informational signs. It is the reason why the application has been developed for mobile phones – people have them always at hand. In short, this application helps people understand text written in Braille characters used by visually impaired or blind people. It helps to break communication barrier between visually impaired and unimpaired people.

The main problem of the whole process is a detection of the dots of a Braille character. Each character consists of one to six dots, which are embossed into paper or other surface. Thus, the dots are little elevated above the background. This provides a way for detection of the dots, because the dots reflect light in a different way than a background. However, certain adjustments need to be done mainly because of the noise in the image. The last main problem is grouping the dots into the particular characters. This needs to be solved by measuring spaces between dots.

This issue is very specific. Only a handful of papers dealing with conversion of text written in Braille exists. Although Braille is used by blind people all over the world, it is quite neglected by the rest of the population. Existing works focus mainly on translating whole pages of Braille, e.g., books. Usually, an image is acquired with a scanner and the image is then processed by a computer. Due to time consuming nature of such task this method is not suitable for interactive mobile application. Our application is capable of detection of dots from images taken by mobile phone, which means the images can be very often inaccurate.

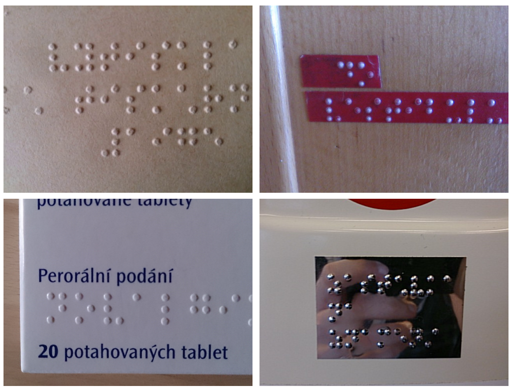


Figure 1. Examples of different texts written in Braille. These photos are taken from books, door signs, captions on drug covers, and informational signs in trains.

Furthermore, to secure a smooth running of the application, the images are taken in low resolution compared to the images taken by scanner. Detailed description of previous works is included below (Sec. 2).

The solution is based on thresholding algorithm, which is quite reliable and successful in detecting dots, yet providing a reasonable amount of frames per second, thus securing optimal running speed of the application. This process is essential, and requires balance between speed of the application and accuracy of output. The algorithm is based on an adaptive method of thresholding. First, an integral image of the original image is calculated. Then, binary image is formed using the integral image. Finally, distances between every single dot are measured, and according to the distance, the dots are grouped into characters. Characters are eventually translated and rendered on screen.

This application works as a pocket Braille reader. It is capable of reading Braille from different surfaces, eg., paper or metal. However, sufficient illumination of the room is required for reliable detection. The speed of application is high enough to secure relatively smooth running. Nevertheless, results of the application are not completely satisfactory if the dots are overlapped with different text or situated in a dark-colored background.

The application was developed for Android mobile platform, focusing on reading only one-sided documents and texts written in Braille. It is available for download at Google Play[1].

1.1 What is Braille

Braille is a unique kind of writing system. It is used mainly by visually impaired or blind people all over the world. It represents a way how can blind people

[1]https://play.google.com/store/apps/details?id=fit.vut.braillereader

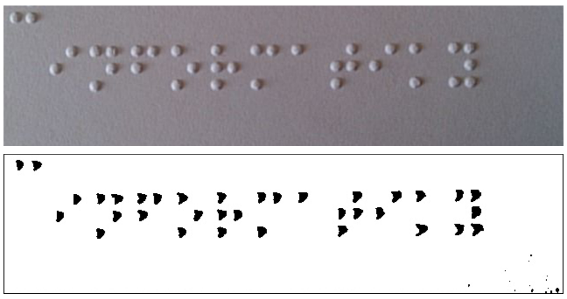


Figure 2. The top image shows the original colored image with Braille. The bottom image shows a binary image after being processed with the thresholding algorithm. The black groups of pixels represent possible dots which have been detected. Some noise is still present.

communicate. Each Braille letter consists of one to six dots, which are aligned into two columns and three rows. These combinations ensure encoding up to sixty-three letters, because empty character is reserved for a space. The dots are embossed into paper or another surface, e.g., metal, allowing people to read the text by touch. Embossing dots requires a special equipment, which means the printing can be expensive, and that is one of the reasons why Braille is not widespread. Some examples of text written in Braille are shown in Figure 1.

1.2 Braille Usage

Although Braille is used all over the world, every language has its own specific character encoding which complicates versatility of Braille. Furthermore, some languages use characters to encode text by syllables or short-cuts, e.g., English. In case of Czech, every letter in the alphabet is encoded separately. Some other characters are encoded as well, e.g., punctuation or mathematical symbols. This ensures that Braille can represent almost any common text. There also exist special prefixes used for encoding numbers and capitals.

2. Overview

This chapter presents several selected works, which deals with optical detection of Braille text. These papers deal mainly with conversion of Braille text on personal computers. Articles were chosen to show different approaches how can Braille be recognized from images. Only a handful of papers deals with this issue on mobile platform.

2.1 Background

The solution published in the article “A Braille Recognition System by the Mobile Phone with Embedded

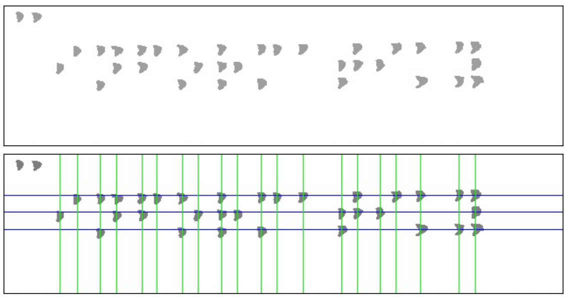


Figure 3. In the top image, the grey spots represent final dots detected by the algorithm. Noise is being removed, and any dots touching edges of the image are being removed as well. The bottom image shows blue horizontal and green vertical lines for dots lying on the same rows and columns. Only one line, i.e., three rows, are preserved, the rest is discarded.

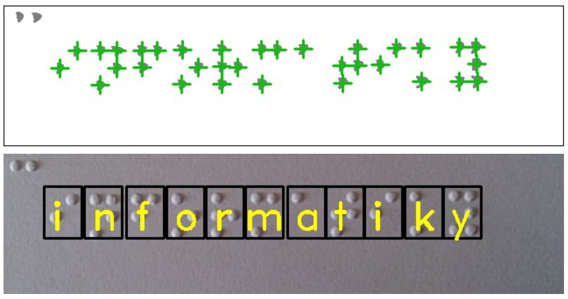


Figure 4. The top image shows a final line of correctly detected dots. Distances between dots are measured, and the dots are grouped into particular characters. The final output of the process is shown in the bottom image. The characters are correctly detected, translated and rendered back on the screen.

Camera" [1] uses a simple thresholding algorithm, converting the original image into a binary image. Sizes of pixel groups are measured, and extremes are considered as noise in the image and thus ignored. The method assumes that dots are perfectly aligned, and does not consider skewed images. In addition, splitting columns of the dots into a character is very simple as well. Only distance between two columns next to each other is measured, and then decided whether the distance is wide enough for splitting columns into two characters or not. The article does not specify any other possibilities, however more complex Braille texts can be very diverse, which means it is important to compare distances between previously detected columns as well, thus it is not possible to rely on this assumption. There are four experimental images mentioned in the article. The average time of image processing is two seconds. Two of the images were correctly translated, one partially, and one not properly translated. There is no further description of the experiments.

The paper "Optical Braille recognition with Haar wavelet features and Support-Vector Machine" [2] processes images with Braille on a computer from a scanned image. The detection algorithm is advanced and complex. It works in a specific sliding window computing Haar wavelets of a small image part. Further, the part of image is processed by Support-Vector Machine method, and considered whether it contains any dot. Using this technique, the image is converted into a binary image containing ones in places of the dots and zeroes in the rest of the image. Although, the correctness of outputs is very good – over 90 %; the process is very time consuming. The tests show that processing one page of text can take up to tens of minutes, which is useless for this project. Also, the correctness of the method depends on hundreds of examples of correct and incorrect images. The geometric correction of skewness of the image is very simple as well. It is solved by manipulation with the original image by drawing a specific rectangle onto paper and then detecting it, which is not very useful for our project as well. The article does not specify any experiments or test sets.

Method used in the article "An Efficient Braille Cells Recognition" [3] is also based on thresholding like other studies. However, the algorithm does not produce a binary image, but a three valued image. It contains black color for background, and white and gray colors for both light and dark sides of each dot. The boundaries (lower and upper) for thresholding are computed on the basis of Beta distribution, which uses histogram of the image for predictions. Grouping of the dots is solved by creating, firstly, a grid for each row on a page. Secondly, by drawing a grid for columns while separating dots on each row. The authors claim that correctness of the process is up to 100 % using this technique. However, this algorithm requires a precise image taken by scanner, and the detection runs on a computer, because it is time consuming as well. The article shows results for several images, however there is no detailed description of the experiments.

The article "Braille document recognition using Belief Propagation" [4] uses Radon transformation for correcting skewness of image, and then uses Belief Propagation technique to detect Braille characters in image. Firstly, image and background are separated. Secondly, Radon transformation is applied with different angle parameters. Using this method, a correct angle is found to rotate image properly, because intensity of image is at its peak at the correct angle. Finally, the image is processed by thresholding, and distances between dots are measured. Further, characters are recognized using Belief Propagation by checking neighboring dots. Success of dots recognition using this technique varies between 88 % up to 100 % according to the experiments. Character recognition rate is almost 83 % on average. The evaluation was carried on several test images.

The technique used in paper “A robust probabilistic Braille recognition system” [5] deals with recognition of dots, which is based on Expectation-maximization algorithm. At first, spacing of the document containing Braille text is measured. Vertical distances between dots and edges of paper are detected. This helps to split the text into lines of Braille. Then, horizontal spacing between edges and columns is measured. Positions of particular dots are modeled as data sequences drown from Hidden Markov process. This method copes well with noise in the image and other artifacts. The processing of one page took up to 14 seconds on testing computer. The success rate of character recognition is almost 100 % according to the several test documents.

2.2 Detection algorithm

The designed algorithm for detection of dots is based on the article “Adaptive Thresholding Using the Integral Image” [6]. The paper presents a reliable form of an adaptive thresholding algorithm, which is usable even for real time image processing. The paper extends Wellner’s algorithm [7] by using an integral image. Computing the integral image requires an additional pass through the image, thus the algorithm is a little bit slower. Further, the algorithm compares value of each pixel to values of surrounding pixels, which brings better image separation. This technique produced the best results in thresholding the image.

3. Detection Process

There are two main factors that has a major impact on the solution – speed and quality. The processing speed has to be fast enough to secure smooth running of the application, and the quality has to be good enough to guarantee highest possible accuracy of the output. These factors need balancing, and it is important to reach a compromise between them, since the hardware of mobile phone is not as efficient as desktop computer hardware.

3.1 Algorithm based on Thresholding using Integral Image

First of all, the original image is converted from colored space into grayscale, which discards unnecessary information about image. We focus on reflected light of the dots, which is better visible on a grayscale image. An integral image is computed in the next step. Following algorithm shows the computation:

Algorithm 1: INTEGRAL IMAGE

```
input  : Input image $in[w][h]$.
output : Integral image $Int[w][h]$.

for $x = 0$ to $w-1$ do
    for $y = 0$ to $h-1$ do
        $out = in[x][y]$;
        if $(x-1 \geq 0)$ then
            $out = out + Int[x-1][y]$;
        end
        if $(y-1 \geq 0)$ then
            $out = out + Int[x][y-1]$;
        end
        if $(x-1 \geq 0$ and $y-1 \geq 0)$ then
            $out = out - Int[x-1][y-1]$;
        end
        $Int[x][y] = out$;
    end
end
```

Where w is width of the image, h is height of the image, out is the output value, in is the original image, and Int is the integral image.

After the calculation of the integral image is complete, a specified sized window is created and moved through the image from left to right. Each pixel in the window is assigned a new value by summing values of surrounding pixels and averaging them. Comparing value of pixel to average values of surrounding pixels in window helps to separate dot from the rest of the image, i.e., it helps to highlight shadows of particular dots. Following formula is used to calculate sum of pixels in the window:

$$\sum_{x=x_1}^{x_2}\sum_{y=y_1}^{y_2} f(x,y) = I(x_2,y_2) - I(x_2,y_1-1) - I(x_1-1,y_2) + I(x_1-1,y_1-1) \tag{1}$$

The sum is calculated for every pixel of the image $f(x,y)$ using values of the integral image I. The x_1, y_1 coordinates represent top-left corner of the window, and x_2, y_2 coordinates represent bottom-right corner of the window.

Following formula is used to create a binary image:

$$g(x,y) = \begin{cases} black & f(x,y) \leq \frac{s\cdot(1.0-p)}{c} \\ white & \text{otherwise} \end{cases} \tag{2}$$

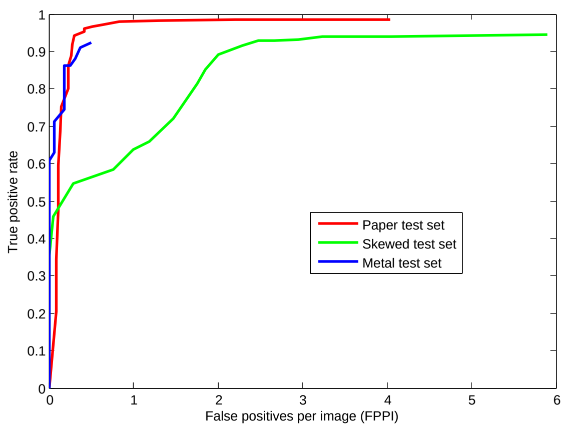


Figure 5. ROC curves representing hit rate for each test set. The curves are created by plotting True positive rate against False positives per image (FPPI). The higher the True Positive Rate, i.e., hit rate, the better success rate. FPPI represents an average number of false alarms per image, i.e., the number of occurrences incorrectly detected as dots.

Where $f(x,y)$ is value of the current pixel, c is number of pixels in the window, s is the total sum of surrounding pixels in the window, and p is tolerance of the sum in percent. The application uses $p = 0.15$ tolerance. Using this method, a binary image is created, containing black color on a possible dot position and white color for the rest of the image. Output of this technique is shown in Figure 2.

The binary image is iterated in the next step, and if a black color occurs, the position is processed using flood fill algorithm looking for its boundaries. Spots too small or large are considered as noise in the image and thus discarded. This technique creates a list of dots detected in the image. The final output of the algorithm is shown in Figure 3.

3.2 Grouping Dots into Characters

First, the algorithm looks for dots in the same row according to their position. This process requires some tolerance for measuring distances, because of the possible skewness of the image. Second, dots in rows are split into columns. If any dot is not assigned with its parent column and row, it is discarded, considered as noise. Finally, characters are created by measuring spaces between the dots and around them. Every character is assigned with columns (left and right) and rows (top, middle and bottom) from which it consists. Every character is assigned a value of the dot, according to its position as well. Eventually, the character is converted into a letter based on its value. The letter is rendered on the screen at the position of the character, afterwards. The result is shown in Figure 4.

4. Experiments

Detection algorithm was tested on three different sets of samples. These sets were named *Paper*, *Skewed*, and *Metal*. *Paper* test set contained images of Braille taken from books and articles only, i.e., the surface was paper. The images were very little skewed and were well illuminated. *Skewed* test set contained only images which were skewed, blurred, taken from wide angles or inaccurate in other ways. *Metal* test set contained various images with Braille embossed into metal surface and usually poorly illuminated. Thus, *Skewed* and *Metal* test sets are mainly experimental. Samples from these sets are included in Figure 6.

Table 1. Detection Rate Statistics

Test set	Recall	Precision
Paper test set	98.17 %	95.17 %
Skewed test set	93.94 %	86.92 %
Metal test set	91.09 %	90.04 %

4.1 Detection Rate

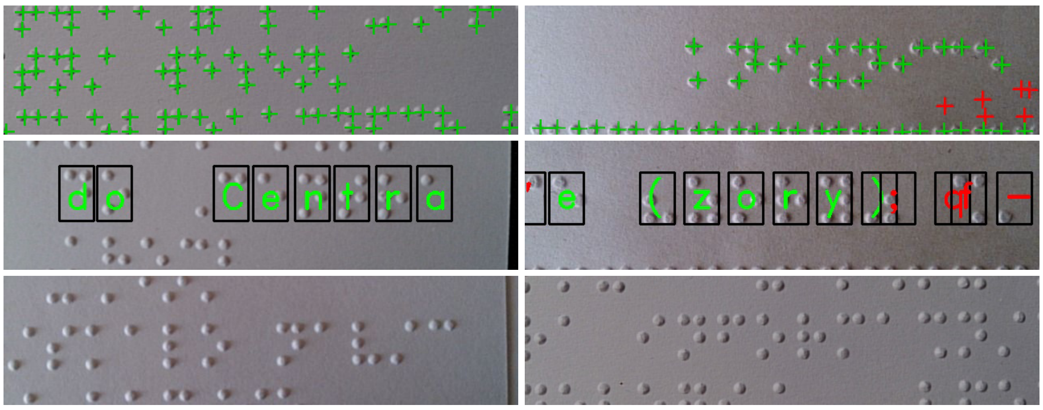
The first part of the evaluation was based on receiver operating characteristic (ROC) curves [8]. The ROC curves express capability of successful detection of the dots. The major measured factors are hits, misses and false alarms. These factors are measured for every image. True positive rate, or recall, represents hit rate of the algorithm. Positive predictive value, or precision, represents probability that an occurrence is truly positive. The corresponding graph is shown in Figure 5. The exact results are listed in Table 1.

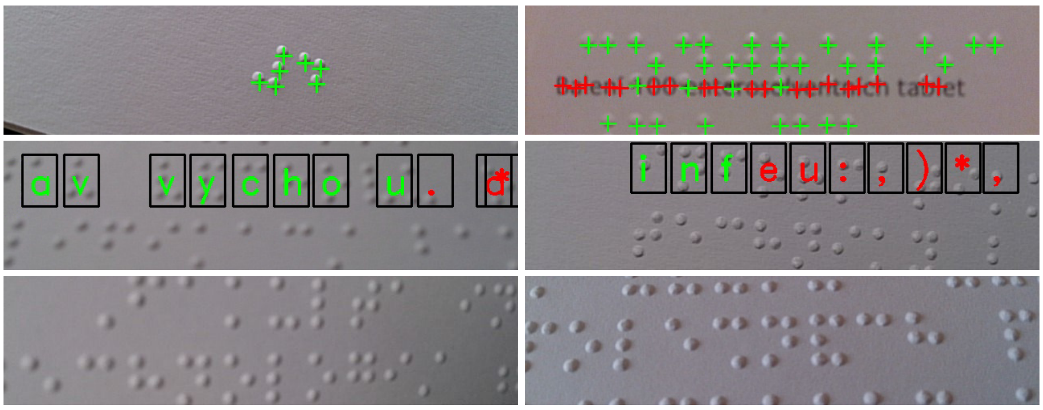
Results show that the hit rate of the detection algorithm is really good. It scored over 98 % in the *Paper* test set, almost 94 % in the *Skewed* test set and 91 % in the *Metal* test set of success rate. The table shows results for value of threshold within values 15 and 195. These numbers express size of dots detected in the image.

4.2 Character Error Rate

The second part of the evaluation was focused on measuring number of correct characters of the output. Measuring Levenshtein distance [9] was used for this purpose. It is metric used for calculating difference between two words. Value of the distance represents how many character adjustments had to be done in order to modify one word into another.

The results of this technique are listed in Table 2. The error rate of the first test set was only 8 %. The success rate of the other test sets was not very successful mainly because of the images of poor quality and their


(a) *Paper* test set


(b) *Skewed* test set

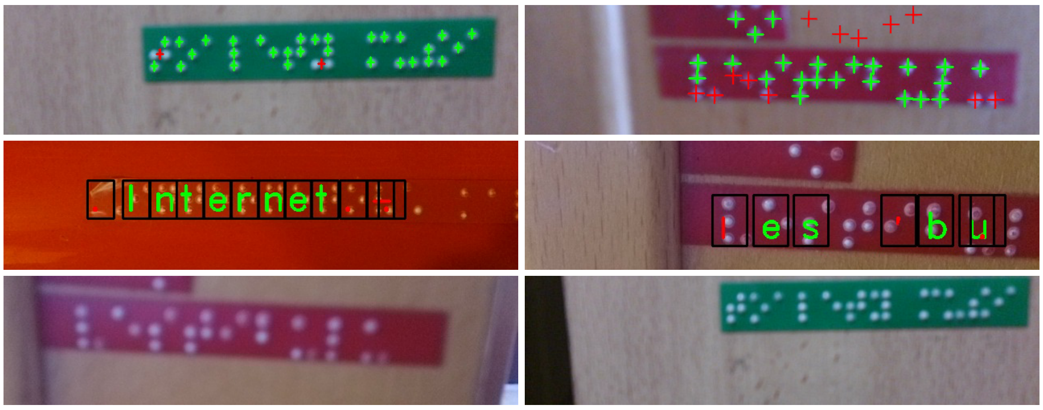

(c) *Metal* test set

Figure 6. Examples of the test sets. The figure shows examples of correctly and incorrectly detected dots in the images, and examples of correctly and incorrectly detected characters from each set. There are also shown raw images from each test set.

Table 2. Character Error Rate

Test set	Error Rate
Paper test set	7.59 %
Skewed test set	46.52 %
Metal test set	59.34 %

skewness. The algorithm still needs to be improved, so it could handle images of bad quality better.

4.3 Running speed

The application was implemented on Android platform using OpenCV[2] library for image processing. The average number of frames per second (FPS) was 13.50 on testing device – Sony Xperia Z, at resolution of 320x240 pixels. However, the average running speed of the application in the same resolution without

[2]OpenCV is an open source multi-platform library used for image processing. Website address: http://opencv.org/

any image processing using the OpenCV for camera handling was about 15 FPS. Thus, the speed of the designed algorithm can be considered as quite good and well optimized. Compared to the other works, processing of one image takes $\frac{1}{13.5}$ of a second. However image processing presented in other works takes up to tens of seconds or even minutes.

5. Conclusions

The paper focuses on creation of a mobile Braille reader. The application provides a method to read Braille with a mobile phone, allowing people to read and understand texts usually used by blind and visually impaired people. Since Braille is a somewhat neglected, it helps to spread its utilization.

The application shows best results in detection of Braille from paper. The hit rate of detection of dots is over 98 % and character error rate is less than 8 %, which makes the application quite reliable. The overall average hit rate is about 94 %. However, mainly character recognition needs improvement when reading Braille from skewed images and images of poorer quality with insufficient illumination. The application was tested on mobile phone with Android platform, and is able to run at 13.50 FPS on average.

This project brings a new method of reading Braille. It is capable of fluent reading of books or signs written in Braille. One of the main advantages of the application is its mobility so it can be used almost anywhere. Another advantage is its simplicity and efficiency. There is no necessary involvement of the user, application translates the text by hovering a mobile phone over Braille characters.

The application can be enhanced by adding support for various language sets, so it could be capable of reading Braille written in any language. This is possible because of the fact that Braille characters are same in every language, only represented variously. Another improvement can be done by enhancing the detection algorithm to work better even in low illumination. This could be possibly done by using flashlight of the mobile phone as an additional source of light. However, using flashlight can be problematic due to handling the camera by the OpenCV. It would require a different implementation of the camera handling and would lead to a possible performance loss. Another problem is that the light is very bright at short distances and could possibly impair input images.

Acknowledgements

I would like to express my gratitude to my supervisor, Ing. Jakub Sochor, for his expertise, understanding, and assistance. His guidance helped me during research and writing of this project.

References

[1] Shanjun Zhang and K. Yoshino. A braille recognition system by the mobile phone with embedded camera. In *Innovative Computing, Information and Control, 2007. ICICIC '07. Second International Conference on*, 2007.

[2] Jie Li, Xiaoguang Yan, and Dayong Zhang. Optical braille recognition with Haar wavelet features and support-vector machine. In *Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010 International Conference on*, volume 5, pages 64–67, 2010.

[3] A.-M.S. Al-Salman, A. El-Zaart, Y. Al-Suhaibani, K. Al-Hokail, and A.-A.O. Al-Qabbany. An efficient braille cells recognition. In *Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on*, pages 1–4, 2010.

[4] Zhenfei Tai, Samuel Cheng, Pramode Verma, and Yan Zhai. Braille document recognition using belief propagation. *Journal of Visual Communication and Image Representation*, 21(7):722–730, 2010.

[5] M Yousefi, M Famouri, Behrooz Nasihatkon, Zohreh Azimifar, and P Fieguth. A robust probabilistic braille recognition system. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(3):253–266, 2012.

[6] Derek Bradley and Gerhard Roth. Adaptive thresholding using the integral image. *Journal of graphics, gpu, and game tools*, 12(2):13–21, 2007.

[7] Pierre D Wellner. Adaptive thresholding for the digitaldesk. *Xerox, EPC1993-110*, 1993.

[8] Tom Fawcett. Roc graphs: Notes and practical considerations for researchers. *Machine learning*, 31:1–38, 2004.

[9] R William Soukoreff and I Scott MacKenzie. Measuring errors in text entry tasks: an application of the levenshtein string distance statistic. In *CHI'01 extended abstracts on Human factors in computing systems*, pages 319–320. ACM, 2001.