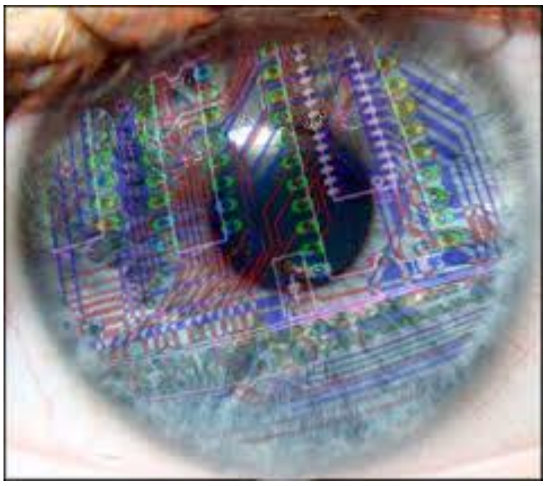
Ben-Gurion University of the Negev

Faculty of Engineering Sciences

Electrical Engineering

Final Report

"Scale Invariant Braille Translator"



Yaniv Tocker, 200095602

Final Project in 'Introduction to Computational & Biological Vision' Course

Submission Date: 19.3.14

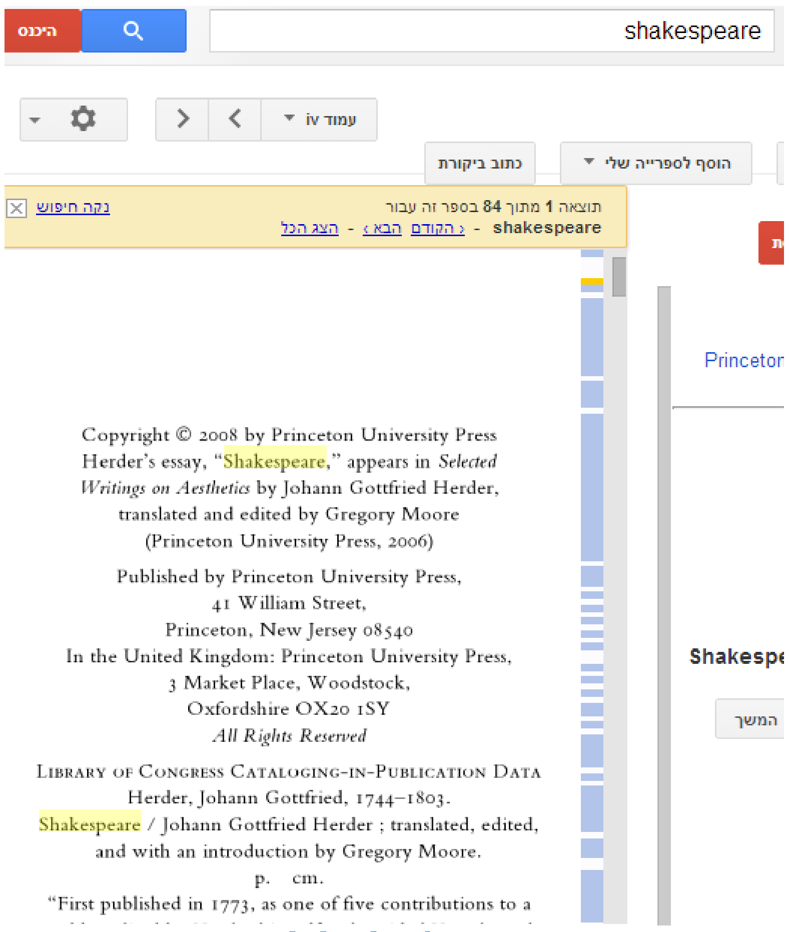
Table of Contents

1. Background

Optical Character Recognition (a.k.a OCR) is the process of translating letter/digits that appear in images and being able to convert them into a format that a computer can manipulate (like strings & ASCII code).

OCR is one of the most active fields in computer vision research since its applications seem to be endless, from automatically reading car plates digits to recognizing street signs in different language.

One popular use of OCR is "Google Books" in which google scans books in order to create electronic copies of them that are researchable for keywords. As can be seen in fig.1 typing the word "**Shakespeare**" leads to a book containing several works by the famous British playwriter.



Copyright © 2008 by Princeton University Press
Herder's essay, "Shakespeare," appears in *Selected Writings on Aesthetics* by Johann Gottfried Herder,
translated and edited by Gregory Moore
(Princeton University Press, 2006)

Published by Princeton University Press,
41 William Street,
Princeton, New Jersey 08540
In the United Kingdom: Princeton University Press,
3 Market Place, Woodstock,
Oxfordshire OX20 1SY
All Rights Reserved

Library of Congress Cataloging-in-Publication Data
Herder, Johann Gottfried, 1744–1803.
Shakespeare / Johann Gottfried Herder ; translated, edited,
and with an introduction by Gregory Moore.
p. cm.
"First published in 1773, as one of five contributions to a

Figure 1 – **Google books** demonstration

1.1. Braille OCR

Using OCR on braille language is a natural application to explore the importance of OCR.

Braille language is a way to let the blind read, while each set of 6 six (3 rows & 2 columns) dots represent a certain letter, therefore words are simply a row of 6 dot block as can be seen in figure 2.

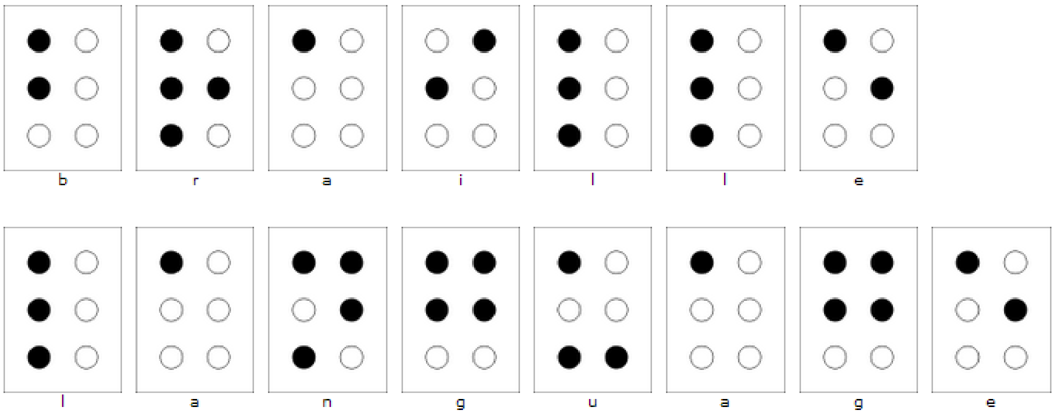


Figure 2 – the words "Braille Language" written in braille language.

In order to assist the blind, braille reading is getting more common in public places such as bus stops. Although braille is used in public the blind don't know that it is there as viewed in figure 3.

Therefore being able to detect & interpret braille language in a scene could be a life-changing ability for the vision impaired. I decided to focus on the 2nd challenge – to interpret braille language to ASCII code.

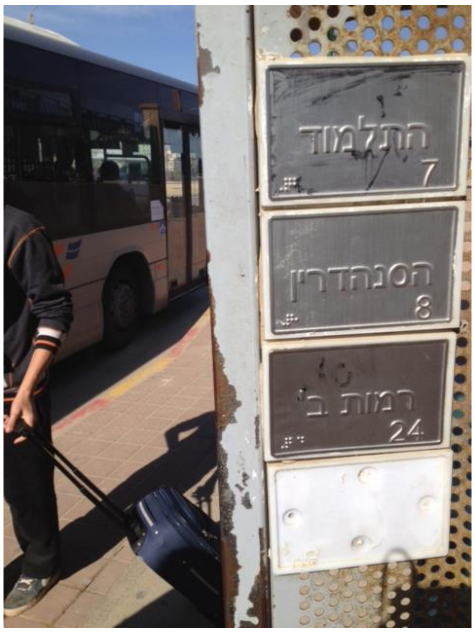


Figure 3 – Braille at a bus stop

A system which does so correctly can be used to teach the blind Braille language and of course to be the 2nd phase in after locating braille writing in a scene.

1.2. **Project Objectives.**

Main goal – to design a software system that receives images containing braille language as input and after processing the image the output will be an image with the letters interpreted to English.

Since braille can appear in different scales, such a system should be able to interpret braille with a variety of circle size options.

In order to achieve the main goal we'll need to accomplish these sub-goals:

- Detect circles in a scene.
- Find the basic braille template – 6 dots arranged in 3 rows & 2 columns.
- Eliminate circles that aren't in the template that was detected in the previous section.
- Interpret the dots to meaningful words and sentences.

2. Methods

2.1. Assumptions

In order to achieve main goal – interpreting braille language from images, we used the common braille template found in images – 6 circles for each letter, as demonstrated in figure 2.

2.2. System Overview

The system works according to the workflow:

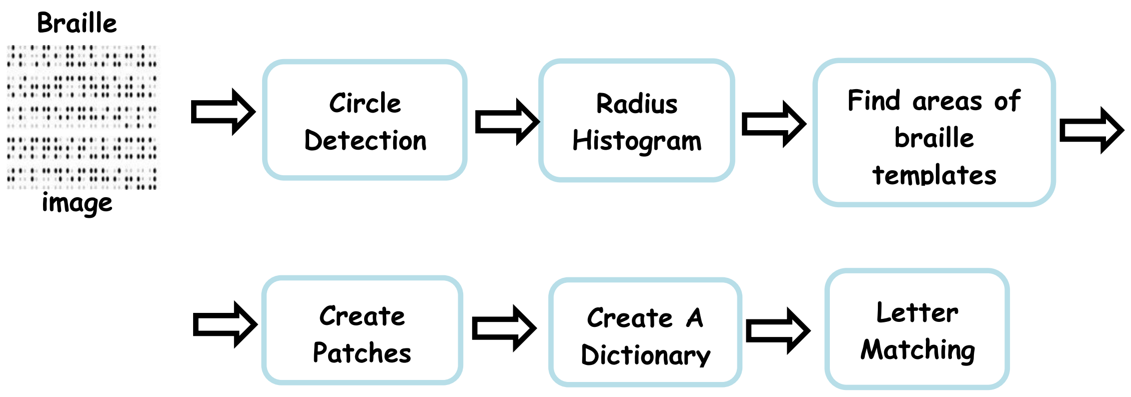


Figure 4 – system workflow

2.2.1. Circle detection

Using Circular Hough Transform (CHT) we we're able to detect circles in the image. Since braille language can appear in many sizes its necessary to being able to detect circles in a large variety of sizes.

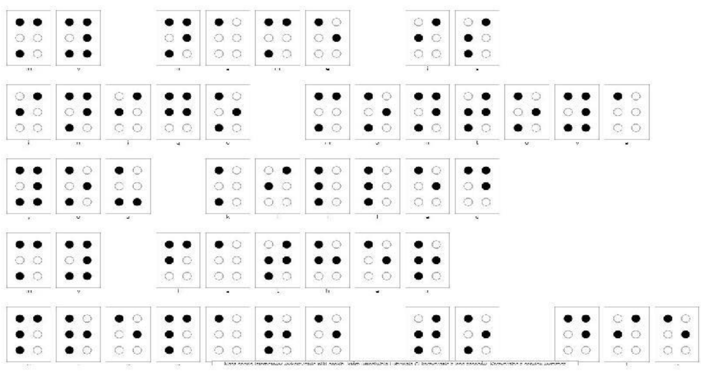


Figure 5 – original image

In most of the

images examined there were circles with radius ranging from 10-125 pixels, therefore CHT was set to find circles with a radius range of 5-30 pixels (this can always be changed in the system settings).

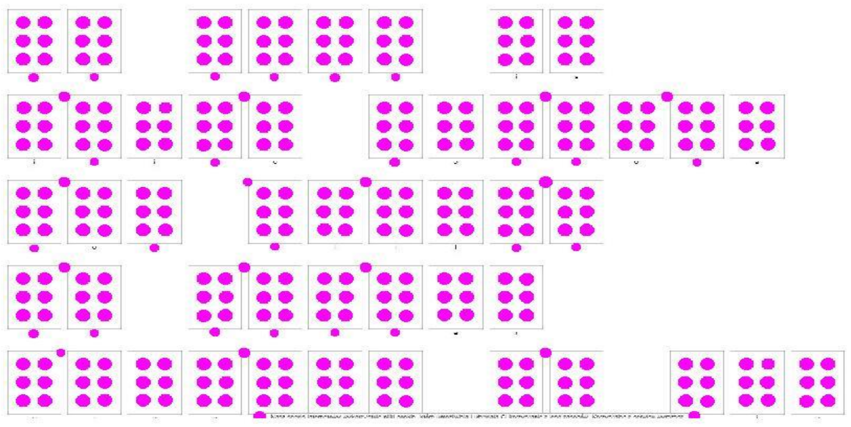


Figure 6 – Circles detected in original image

2.2.2. Radius Histogram

In order to achieve a system that is robust to circle size, it is necessary to design it *dynamically*. This required characteristic is achieved by finding the most common radius of circles in the image.

This feature will be explained more in sections 2.2.3 & 2.2.5.

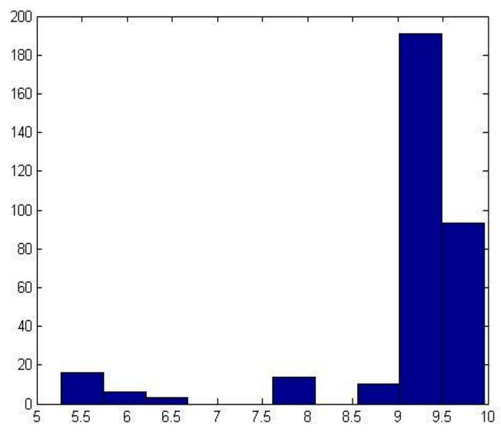


Figure 7 – radius histogram

Acquiring the radius histogram allows us ignore detected circles in the image that their radius is far from the most common one, as demonstrated in figure 8.

In order to also eliminate detected circles in an image that don't belong to the 6-dot template in braille language, each circle was searched for 'closest neighbors'.

If a circle didn't have at least 3 circles that their center is less than 4 times the common radius they were erased as seen in figure 9.

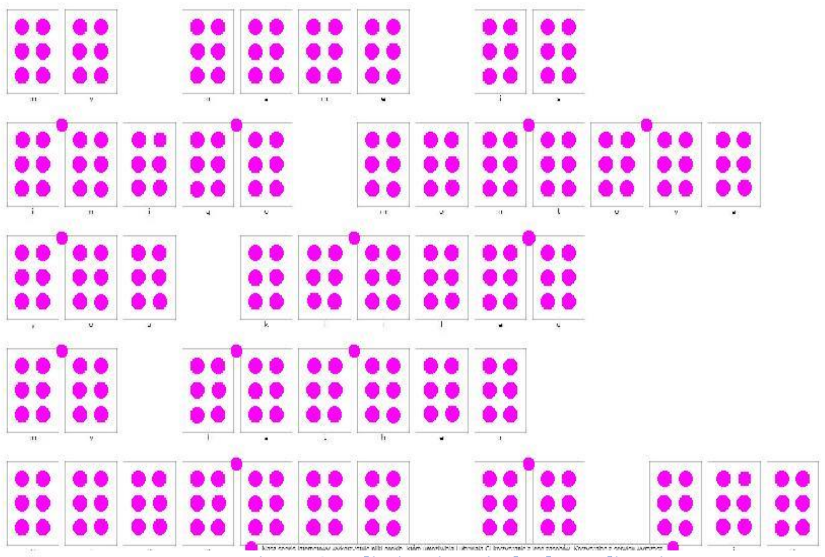


Figure 8 – eliminating circles by radius size

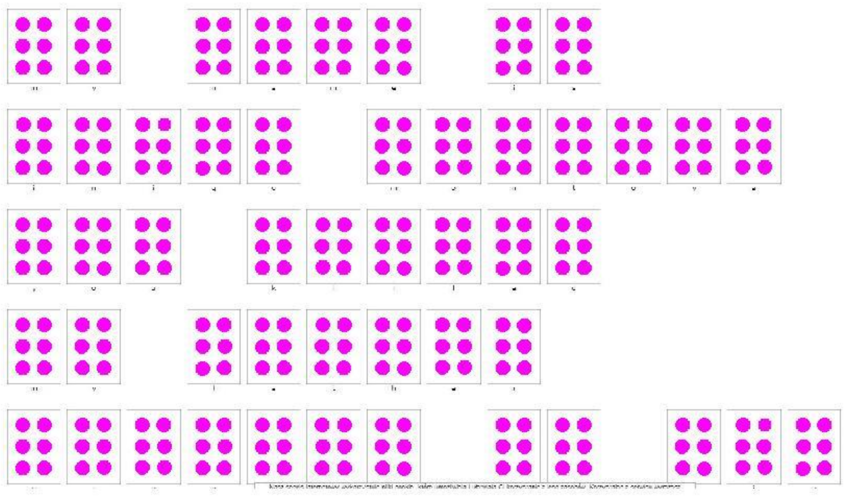


Figure 9 – eliminating circles of 3-nearest neighbors

2.2.3. Find areas of braille templates

After acquiring an image at the previous stage, we would like to find segment the image into the 6-dot template. In order to do so, we found the center of mass for each 6 dots in that fit the template by following these simple steps:

1. A mask of the detected circles was created – a binary image with zeros at the background and ones if a pixels belong to a circle as shown in figure 10.

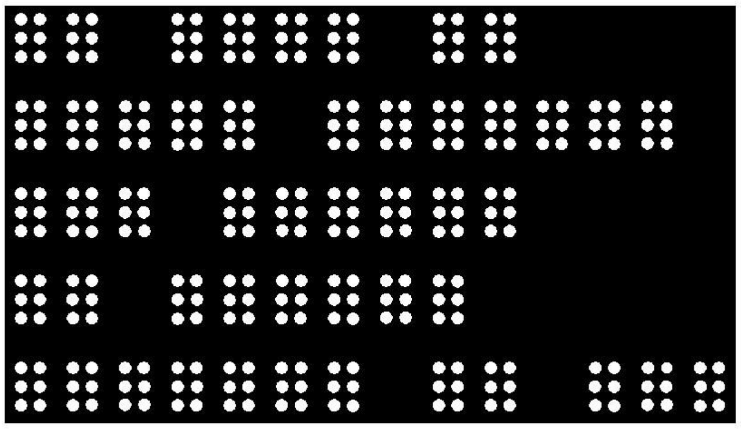


Figure 10 – figure 10 circle mask.

2. A dynamic patch (dynamic since its size is based on the common radius in the image) was created in the size of 9*common radius X 6 times the common radius and 6 circles were placed as seen in figure 11: these dimensions were selected after trying many values, and the chosen ones yielded the best results.

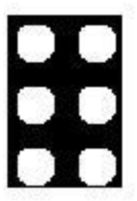


Figure 11 – an artificial patch

3. Template center detection – by applying the idea behind *'Hough transform'*, which is asking 'how many pixels agree that they belong to a template?", and taking local maxima as detections.

   Since the patch created in the previous section is symmetric in both axis, using convolution should to the trick.

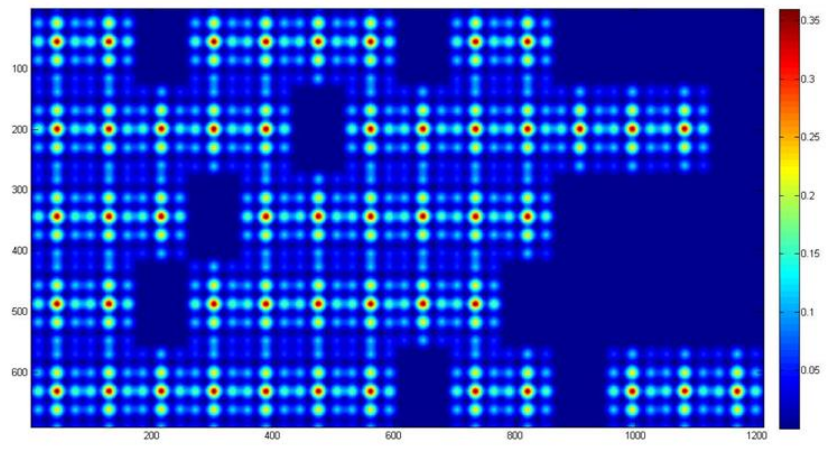


Figure 12 – the result of convolving figures 10 & 11

2.2.4. Create patches

A patch is a part of an image. We would like to create patches where each one contains information needed to interpret one letter – 6 dots.

In order to achieve this the following steps were taken:

1. Create a mask as in section 2.2.3.1 for the original image with only braille 'meaningful' circles – circles that are black.
2. Mark on the mask the 'center of letters' as found in section 2.2.3.3. these centers are considered to be the 'center of mass' for each patch.
3. Create patches with the same dimensions used in section 2.2.3.2 around each center of mass.

These steps are shown in the following figure:

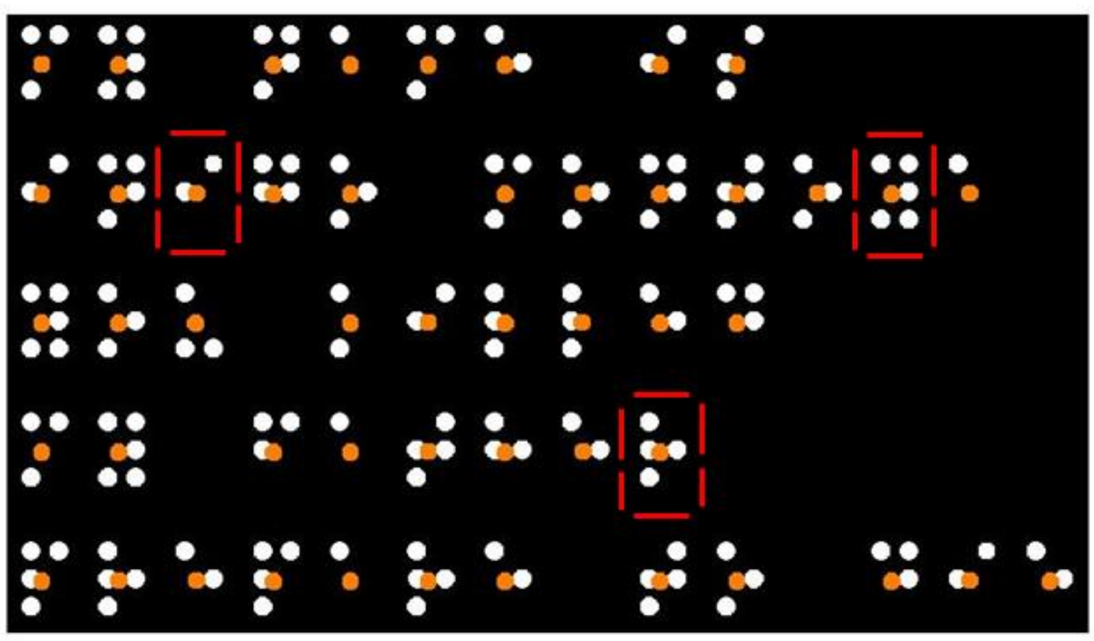


Figure 13 – black & white pixels – the mask, orange circles – centers of mass, red squares – some patches

2.2.5. Create a dictionary

Creating a dictionary is crucial for interpreting braille writing into English letters.

A 'Dictionary' is basically an image-adjusting representation of english letters in braille language.

In order to keep our system robust to scale of braille writing, the dictionary is built for **each image**, this is done by using the size of the common radius to construct it. The size of each letter braille representation n according to the patch declared in section 2.2.3.2. this step is important since it forces the dimensions of the patches to be the same as in the dictionary.

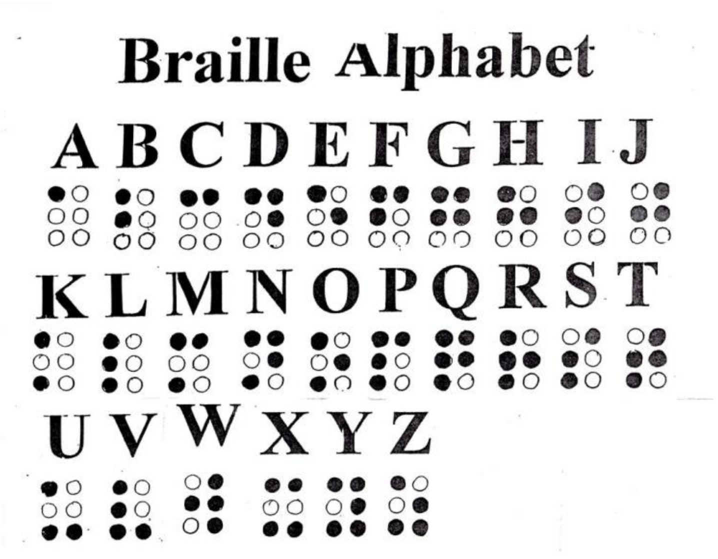


Figure 14 – Braille letters in English

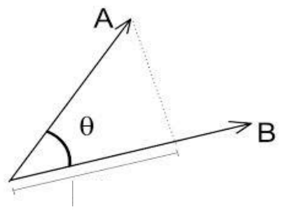
2.2.6. Letter Matching

Letter matching is the final processing step.

It simply takes each patch as detected in section 2.2.4. and compares it to the dictionary created in the previous section to decide which letter the patch is.

The action of letter matching is performed in these following steps:

1. Turn the patch and all of the letter representation in the dictionary to vectors.
2. Calculate the normalized inner product of the patch with **each** letter representation from the dictionary.
3. Classify the patch as the letter which its inner product gave the maximum value.



$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Figure 15 - Normalized inner product theorem

An example of performing these steps is shown in fig. 16:

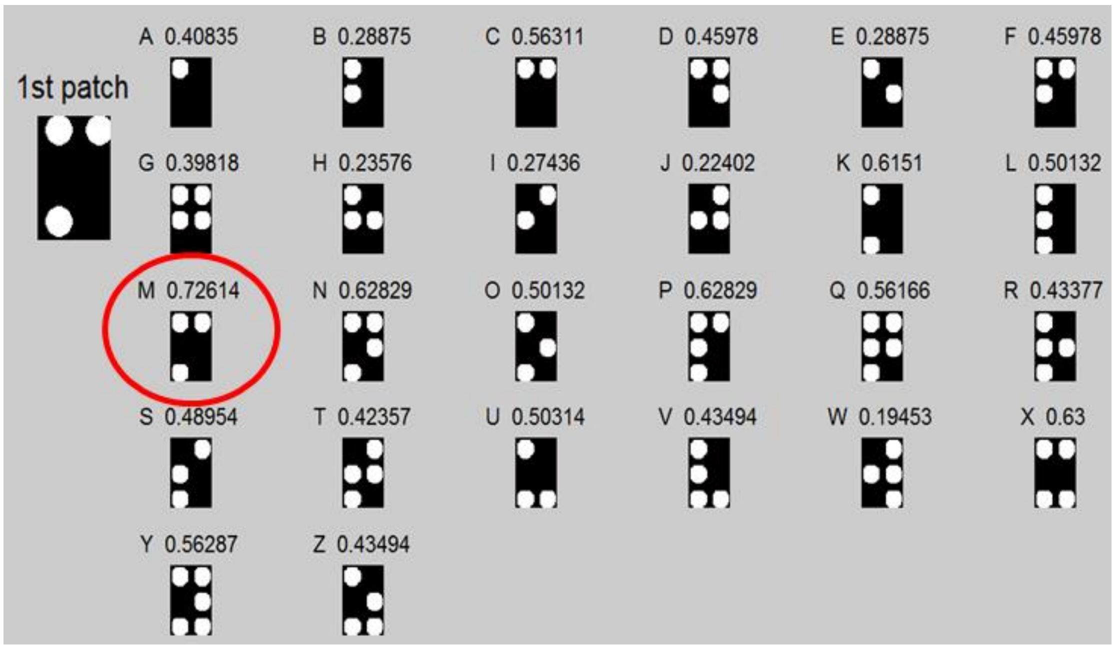


Figure 16 – applying the 3 phases on the patch in the left of the image.

on the image left side on the image is the checked patch. The other squares are the letter representations in the created dictionary. Above each braille letter is indicated the english letter & its inner product result with the checked patch.

The letter representation in the red circle got the highest value, therefore the patch was classified to be 'M'.

Applying these steps in each patch will yield an interpret of the whole image. The result of applying this on figure 5 is shown in figure 17.

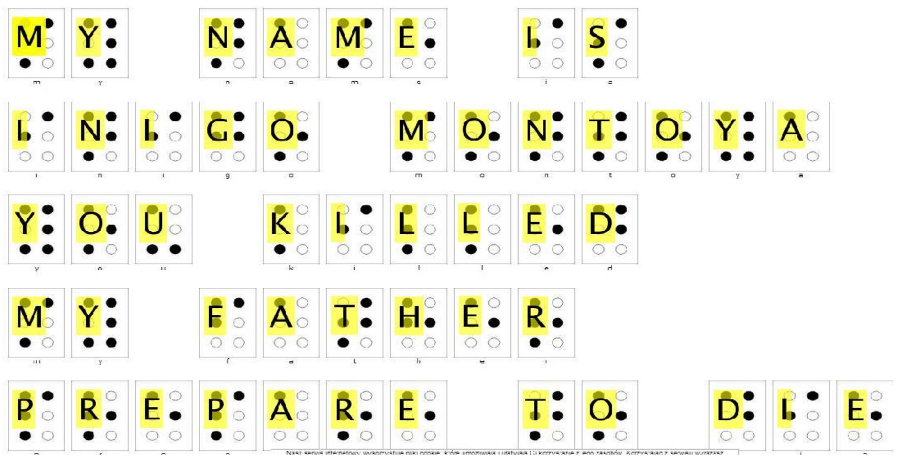


Figure 17 – the interpretation of figure 5

3. Conclusions.

- A novel way to analyze braille language has been proposed.
- The suggested system is robust to scale changes, which is a required skill from such a system since braille can appear in varying sizes.

4. Future work.

- As explained in section 2.1. the system is designed to translate braille writing that appear in the most common way that images of braille are available online- each letter is represented with 6 dots. Not always braille appears this way, since the circles that aren't 'braille meaningful' (white circles) don't always appear. The system isn't design to deal with this scenario, but it is a great platform and lays down the foundations to develop such a system.
- Future development could be to be able to detect braille in a scene from images (like in figure 3) as a prior phase to the suggested system, which could truly make this system a life changing feature for the vision impaired

5. References.

1. OCR: http://en.wikipedia.org/wiki/Optical_character_recognition

2. Braille OCR

http://en.wikipedia.org/wiki/Optical_braille_recognition

3. Braille OCR example

http://www.ni.com/white-paper/6470/en/

4. Braille code generator

http://braille.compelo.com/generate/

5. X. fernanadez et al, “A braille O.C.R for the blind”

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.7727&rep=rep1&type=pdf

6. J.Mennans et al,”Optical recognition of braillle writing using standard equipment”.

http://bauhaus.ece.curtin.edu.au/~iain/PhD%20BU/A_Phd%20docs/To%20read/Accessibility%20info/Research/Braille_Articles/OCR%20of%20Braille.pdf

7. O. ben-shahar – lecture notes from ICBV 2014 –

- object classification

- hough transform

6. Graphical User Interface Manual (GUI)

A GUI was created to simplify the use of the software.

It is capable of interpreting every image with braille writing that obeys the assumptions stated in section 2.1.

A nice braille writing converter from English can be found in the following link: http://braille.compelo.com/generate/

At the website you can create braille writing with whatever you would like to write. Saving it as an image is simple by using windows 'snipping tool'.

The opening screen of the GUI has two buttons – Load & Translate.

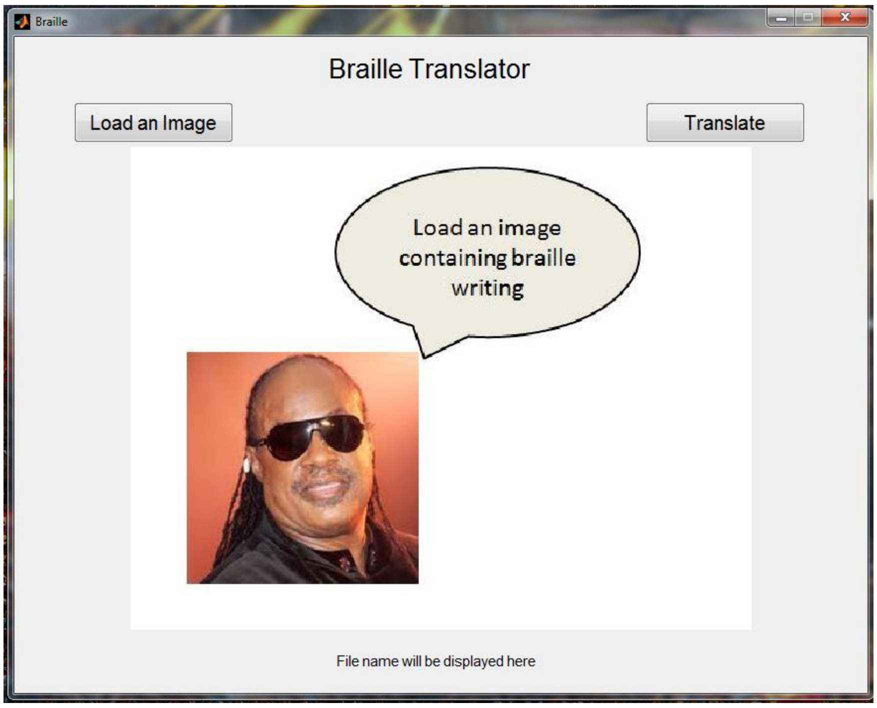
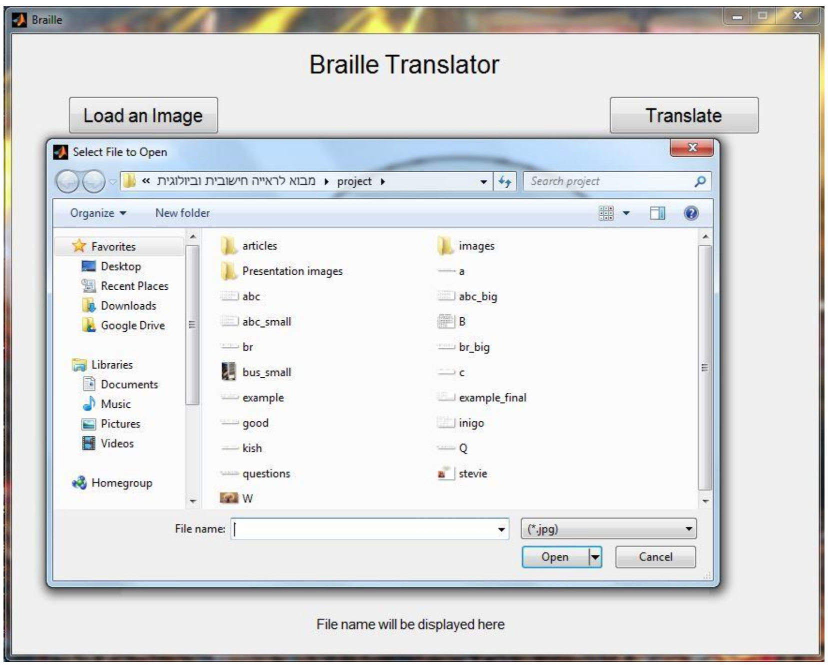

Braille
Braille Translator
Load an Image
Translate
Load an image containing braille writing
File name will be displayed here

Figure 18 – GUI opening screen

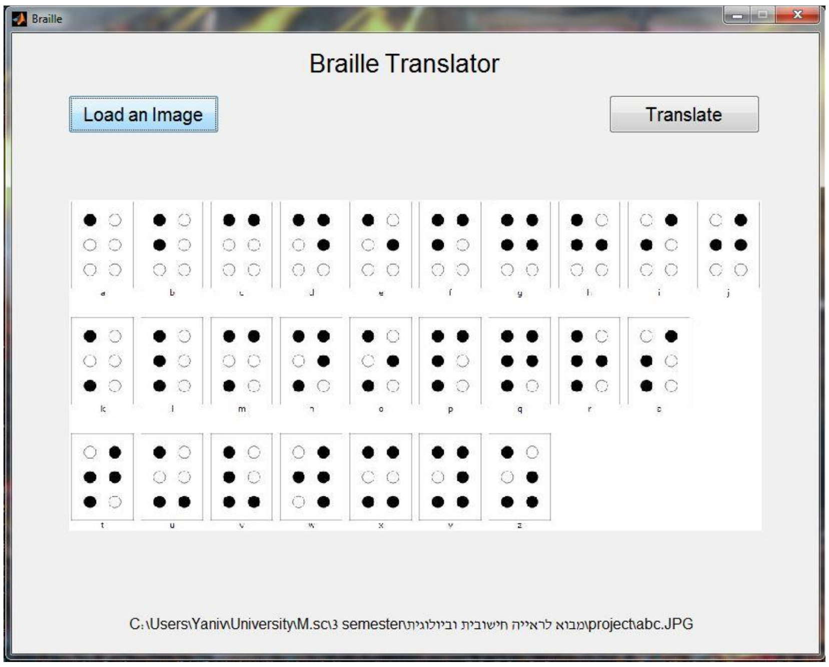
Pressing on 'load an image button' will result in the following window:



File name will be displayed here

Figure 19 – GUI manual – choosing a file

After picking an image it will be displayed in the middle of the GUI:



C:\Users\Yaniv\University\M.sc\3 semester\מבוא לראייה חישובית וביולוגית\project\abc.JPG

Figure 20 – GUI manual - viewing the chosen file

Pressing the 'Translate' button will make English letters appear upon the braille writing:

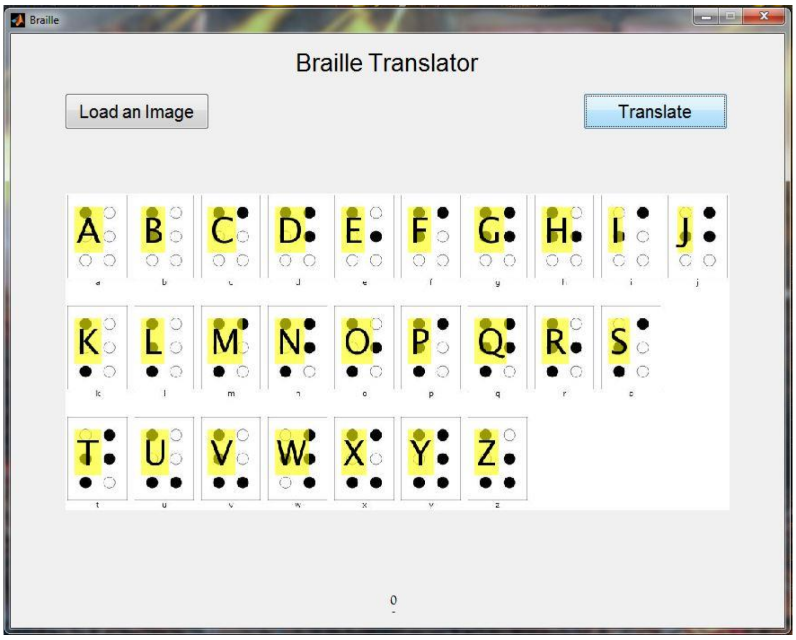


Figure 21 – GUI manual – the result