

## Digital Image Braille Character Recognition, Extraction & Translation System [DIBCRETS]TACT-EYE

\*RushilShah<sup>a</sup>, Harsh Jha<sup>b</sup>, Rohaan Advani<sup>c</sup>

(a: student, Elpro International School, b: ex-student, Elpro International School, c: student, College of Engineering Pune)

Corresponding Author: \*RushilShah<sup>a</sup>

### -----ABSTRACT-----

Braille is a tactile form of reading and writing for the visually impaired, developed by Louis Braille that uses a system of raised dots. Each Braille "cell" or character contains a combination of up to 6 dots. The paper addresses different approaches used to read, recognize, and translate Braille to English via Optical Dot Recognition algorithms using machine learning and Python. We propose a Digital Image Braille Character Recognition, Extraction & Translation System [DIBCRETS], also known as TACT-EYE, which has been tested and can be used by volunteers at NGOs and other organizations for the blind with limited training.

**KEYWORDS:** BRAILLE, OPTICAL RECOGNITION, TRANSLATION, MACHINE LEARNING, PYTHON

Date of Submission: 07-11-2020

Date of Acceptance: 20-11-2020

### I. INTRODUCTION

Braille is a tactile form of reading and writing for the visually impaired, developed by Louis Braille. Visually impaired people are an integral part of society and they, like everyone else, can play an effective role in its development. Therefore, society as a whole must work together to help uplift the blind community. The minds of people who are impaired in one or more of their senses form new connections and enhance one or more of their other senses, for example visually impaired people usually have a compensatory heightened sense of hearing, smell, and touch as well as cognitive functions. The most common communication system for visually impaired people is the Braille system which depends on the sense of the touch, it allows them to read and write using a series of raised dots. People who have their vision intact can read braille with their eyes or via touch.<sup>[1]</sup>

Braille is more of a code than a language by which many languages—such as English, Spanish, Arabic, Chinese, and dozens of others—may be written and read. Braille is used by thousands of people all over the world in their native languages and provides a means of literacy for all.<sup>[2]</sup>

Braille is a system of raised dots, each Braille "cell" or character consists of two columns with three dots, each character is represented by a combination of dots raised at any combination in the six dot "grid". There is 1 possible combination for 0 dots, 6 possible combinations for 1 dot, 15 possible combinations for 2 dots, 20 possible combinations for 3 dots, 15 possible combinations for 4 dots, 6 possible combinations for 5 dots, and 1 possible combination for 6 dots. ( ${}^nC_r = n! / (r!(n-r)!)$ ) or  $2^n$  where n is the number of possible dots. Braille can create more symbols with the combination of two of these "cells"

The Braille dot dimensions are suitable for the tactile resolution of the fingers of the reader. The nominal height of braille dots should be 0.019 inches [0.48 mm] and is uniform within any given transcription, the nominal base diameter of a braille dot is 0.057 inches [1.44 mm]. the Horizontal Distance (HD) and Vertical Distance (VD) between the centers of the dots in the same cell is 0.092 inches [2.340 mm]. The nominal distance from center to center of corresponding dots in adjacent cells is 0.245 inches [6.2 mm]. Approximately, the distance between dots on neighboring cells is 0.15 inches [3.75 mm] horizontally.<sup>[3]</sup>

There are currently 3 Grades of braille, each increasing in the amount of abbreviation. Grade 1 Braille refers to representing only the letters in the alphabet. A combination of these letters makes up words. Grade 1 Braille is less commonly used in books and documents as it is not space efficient. The Grade 2 Braille System emerged as the more space-effective alternative, in which common words are represented with abbreviated forms such as "but", "can", and "do". These abbreviations are standardized. The most recent and space-effective Grade of Braille is Grade 3.<sup>[4]</sup>

This paper attempts to discuss the steps we used in order to create our Digital Image Braille Character Recognition, Extraction & Translation System [DIBCRETS] alternatively referred to as TACT-EYE (Tactile language + Detection of character from a Camera image). Using our system Braille text can be translated into English or any other language via camera image. Our system has been tested and can be used by volunteers at NGOs and other organizations for the blind with limited training.

## II. IMPORTANT TERMINOLOGY

- TensorFlow is an open-source library for numerical computation and large-scale machine learning.
- Keras is an open-source neural network library written in Python which can run on top of TensorFlow
- Haar Cascade is a machine learning-based approach in which the classifier is trained with a large number of positive and negative images.
- Positive images – Contain the images which we want our classifier to identify.
- Negative Images – Images of anything that we don't want our classifier to identify
- GUI (graphical user interface) is a system of visual components for the computer software. GUI represents objects that convey information. These objects also have certain attributes, and when an action is taken by the user these attributes (color, size, or visibility) may change
- Single Shot detector takes only one shot to detect multiple objects present in an image using multibox. It is a fast in speed and high-accuracy object detection algorithm.<sup>[5]</sup>
- NumPy is a Python library, that provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- Canny Edge Detection: Canny edge detection is a multi-step algorithm that can detect edges and suppress noise simultaneously

## III. PROPOSED METHODS

Here are the Proposed Methods to Optically Read, Recognize and Translate Braille to English

### **Method 1: Identify Dots to create a matrix for character identification**

Our first method involved the following 2 main steps

1. AI model trained for detecting the number of dark circles.
2. Using the number of circles detected what alphabet it is. For example, 1 dot is alphabet A, 2 dots is alphabet B, and so on.

#### Detailed Steps

1. The code starts with calling the image clicked by the user after doing the perspective transform.
2. Then our algorithm processes the image through several filters.
  - I. We applied `np.zeros` code to eliminate the background and change everything to black color except the boundaries of darkened dots.

Numpy zero logic:

**`numpy.zeros(shape, dtype=float, order='C')`**

Return a new array of given shape and type, filled with zeros(0,0,0)

And (0,0,0) represents black color.

#### **Parameters**

***shape*** *int or tuple of ints*

Shape of the new array, e.g., (2, 3) or 2.

***dtype*** *data-type, optional*

The desired data-type for the array, e.g., **`numpy.int8`**. Default is **`numpy.float64`**.

***order*** *{'C', 'F'}, optional, default: 'C'*

Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

#### **Returns**

***out*** *ndarray*

An array of zeros with the given shape, dtype, and order.

II. Next, we used canny edge detection.

Canny Edge Detection is a popular edge detection algorithm. Canny edge detection helped our algorithm in Noise Reduction and find the Intensity Gradient of the Image.<sup>[6]</sup>

III. Now comes the contour function. We used this function for detecting the boundary of the dots of the braille character.

Contours can be defined as a curve joining all the continuous points (along a boundary), having the same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

IV. We then passed `cv.CHAIN_APPROX_NONE` which helped us to store all the boundary points of the target object.

V. Using contour hierarchy.

What is Hierarchy?

Usually the `cv.findContours()` function is utilized to distinguish objects in an image. Sometimes objects are in different locations and at times, a few shapes are inside different shapes, as nested figures. In this way, contours in an image have some relationship to each other. Furthermore, we can indicate how one shape is associated with one another. Interpretation of this relationship is known as the Hierarchy.<sup>[7]</sup>

Using the contour method and its attributes we tried getting the boundary of our dots with more accuracy in case our algorithm faces confusion for selecting the boundary between the outer or inner part. Here we used `cv2.RETR_EXTERNAL` which selects the boundary of outer parent circles.

Matrix logic behind `cv2.RETR_EXTERNAL`:

```
>>> hierarchy
array([[ [ 1, -1, -1, -1],
        [ 2,  0, -1, -1],
        [-1,  1, -1, -1]])])
```

Fig 1.1 <sup>[7]</sup>

Using an if-else loop we taught AI to judge the alphabet depending on the no. of dots present in the image. Like 1 dot is alphabet A, 2 dots is alphabet B, and so on

`cv2.drawContours` method was used just to check the accuracy of the boundary being detected by our algorithm.

Using hierarchy, we can go into each individual database and recognize the characters based on the dot position in the matrix of 3x2



Fig 1.2: Sample Braille Cell with assigned numbers

The dot areas with empty space assigned a value 0 and filled being assigned value 1.

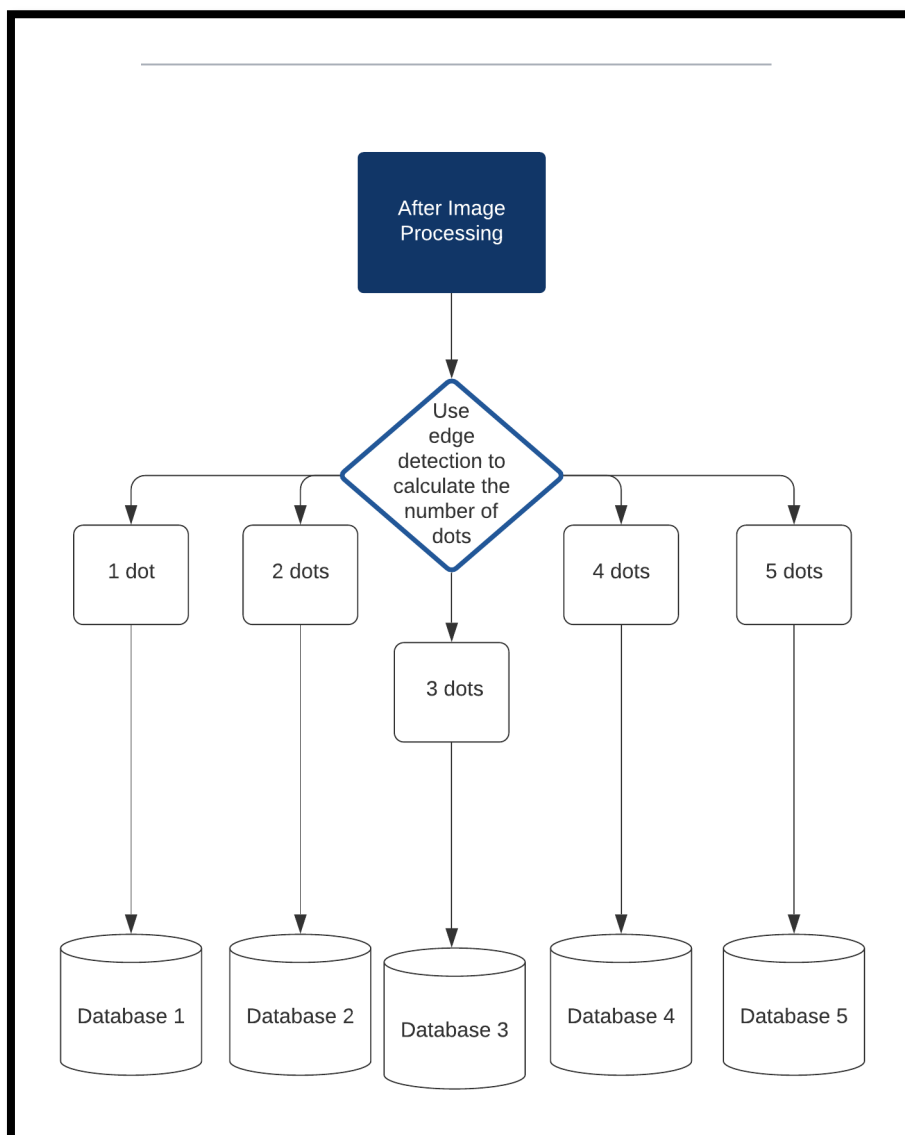


Fig 1.3: Segregating Braille characters based on no. of dots using edge detection

Database 1	A
Database 2	B,C,E,I,K
Database 3	D,F,H,J,L,M,O,S,U
Database 4	G,N,P,R,T,V,W,X,Z
Database 5	Q,Y

CHARACTER	BRILLE	BRILLE DOTS CO-ORDINATES (FIG 1.X)	NUMBER OF DOTS (PART OF DATABASE)
A	·	1	1
B	⋮	12	2
C	⋯	14	2
D	⋮⋮	145	3
E	⋮·	15	2
F	⋮⋮	124	3
G	⋮⋮	1245	4
H	⋮⋮	125	3
I	·⋮	24	2

J	::	245	3
K	:	13	2
L	:	123	3
M	::	134	3
N	::	1345	4
O	::	135	3
P	:	1234	4
Q	:	12345	5
R	:	1235	4
S	:	234	3
T	:	2345	4
U	::	136	3
V	:	1236	4
W	::	2456	4
X	::	1346	4
Y	::	13456	5
Z	::	1356	4

[8]

**METHOD 2: 2 Layered Convolutional Neural Networks**

- A **Convolutional Neural Network (CNN)** is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNNs have the ability to learn these filters/characteristics.

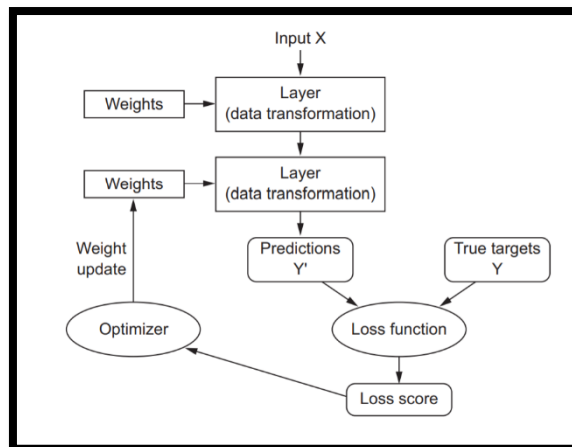


Fig 1.4: Showcasing the Backpropagation algorithm

This attempted to train our AI using a 2 layered CNN model with Tensorflow and Keras.

This method involved the following steps:

- Segment all the collected images in different folders. Each folder contains images of that respective alphabet.

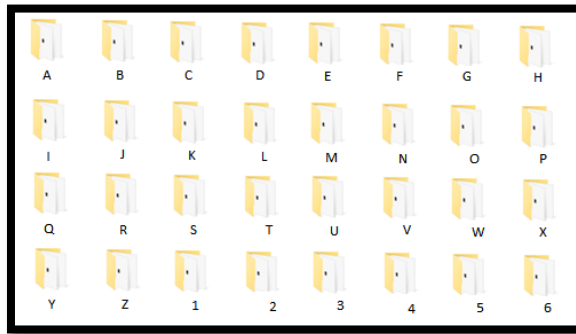


Fig 1.5: Folders containing sample images of the individual characters

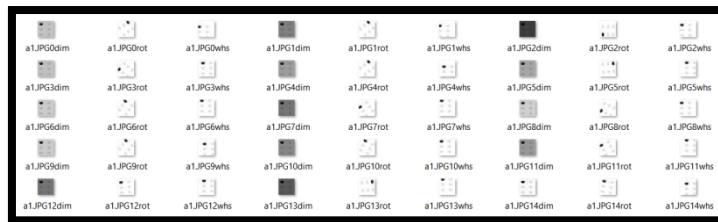


Fig 1.6: Folder A: Contains 63 images of alphabet A

- Now we started training our CNN using this dataset with a different folder

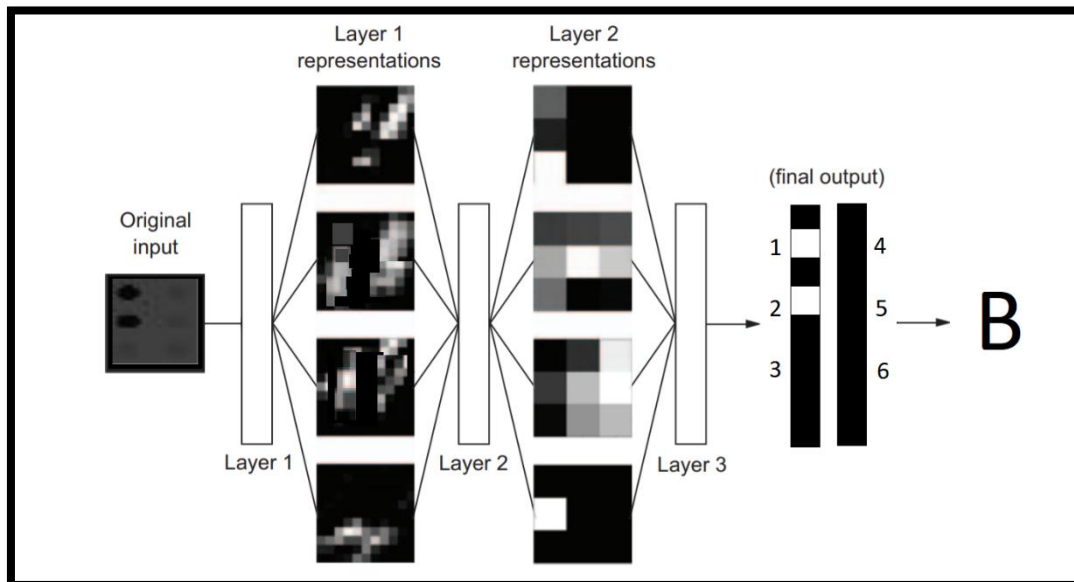


Fig 1.7: Deep representation learned by braille-character classification model.

\*Method failed due to insufficient data for the databases created

\*Method was kept on standby for a more efficient method

### **Method 3: Single Shot Detection Method**

Using the dataset of Method 2, we build our own custom haarcascade file, for which we used Cascade Trainer GUI. The aim was to build a cascade file and run detection algorithms like face/object detection. Creating a cascade file was successful. Now for the recognition, we tried SSD (Single Shot Detection Method). This method again failed because of a lack of positive and negative images of the braille character. AI was able to translate to a limit but on the real-life application, it ran into many issues.



Fig 1.8: Positive Images: Nearly 1560, 28x28 Gray images of Braille Characters



Fig 1.9: Negative Images: MNIST Dataset along with many other English alphabet datasets

#### **Method 4 – DER (Detection, Recognition, and Execution) Method**

Method 4 is a combination of attempt 1 and 2

We will be using a computer-generated braille character paper to demonstrate the process.

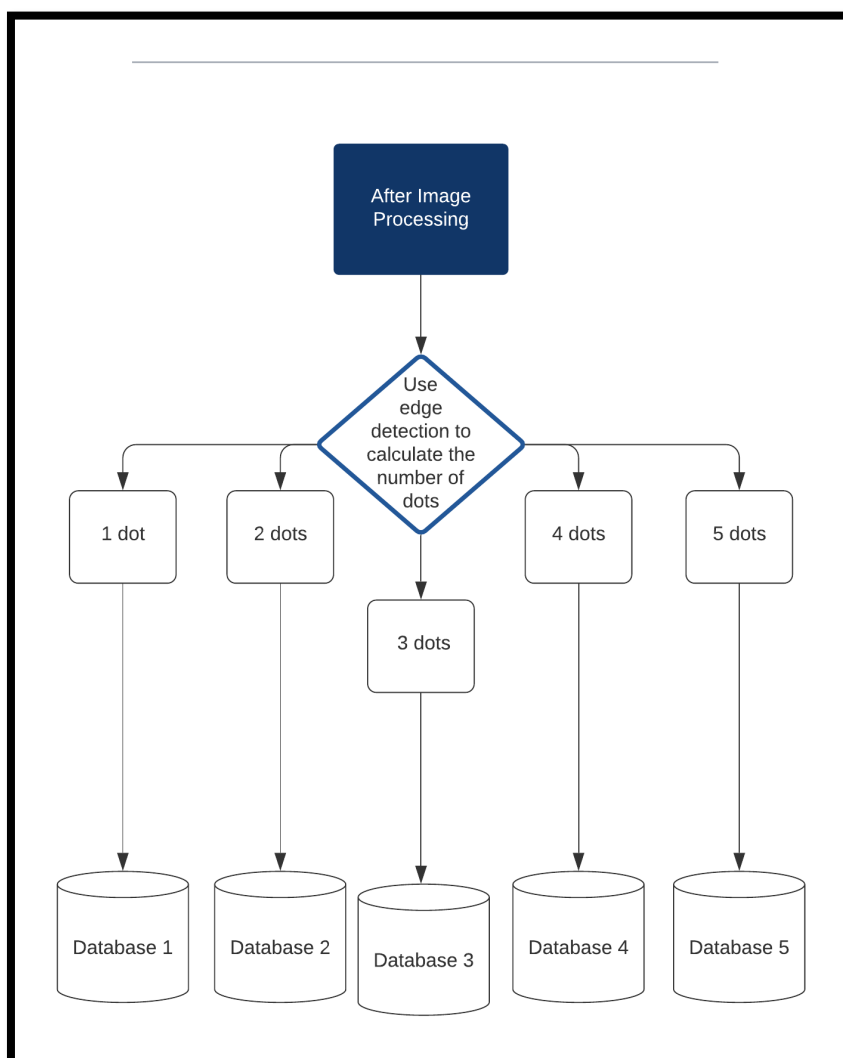


Fig 1.3: Segregating Braille characters based on no. of dots using edge detection

Database 1	A
Database 2	B,C,E,I,K
Database 3	D,F,H,J,L,M,O,S,U
Database 4	G,N,P,R,T,V,W,X,Z
Database 5	Q,Y

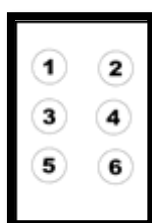


Fig: 1.2  
The empty space being labeled 0 and filled being labeled 1

CHARACTER	BRAILLE	BRAILLE DOTS CO-ORDINATES (FIG 1.X)	NUMBER OF DOTS (PART OF DATABASE)
A	·	1	1
B	:	12	2



C	⠠	14	2
D	⠡	145	3
E	⠢	15	2
F	⠣	124	3
G	⠤	1245	4
H	⠥	125	3
I	⠦	24	2
J	⠧	245	3
K	⠨	13	2
L	⠩	123	3
M	⠪	134	3
N	⠫	1345	4
O	⠬	135	3
P	⠭	1234	4
Q	⠮	12345	5
R	⠯	1235	4
S	⠰	234	3
T	⠱	2345	4
U	⠲	136	3
V	⠳	1236	4
W	⠴	2456	4
X	⠵	1346	4
Y	⠶	13456	5
Z	⠷	1356	4

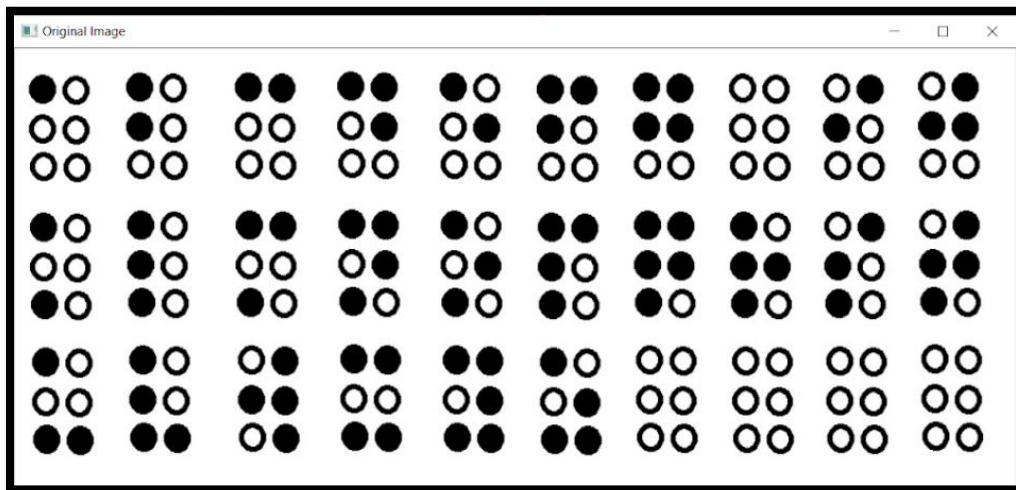


Fig 1.10: Computer generated braille character paper.

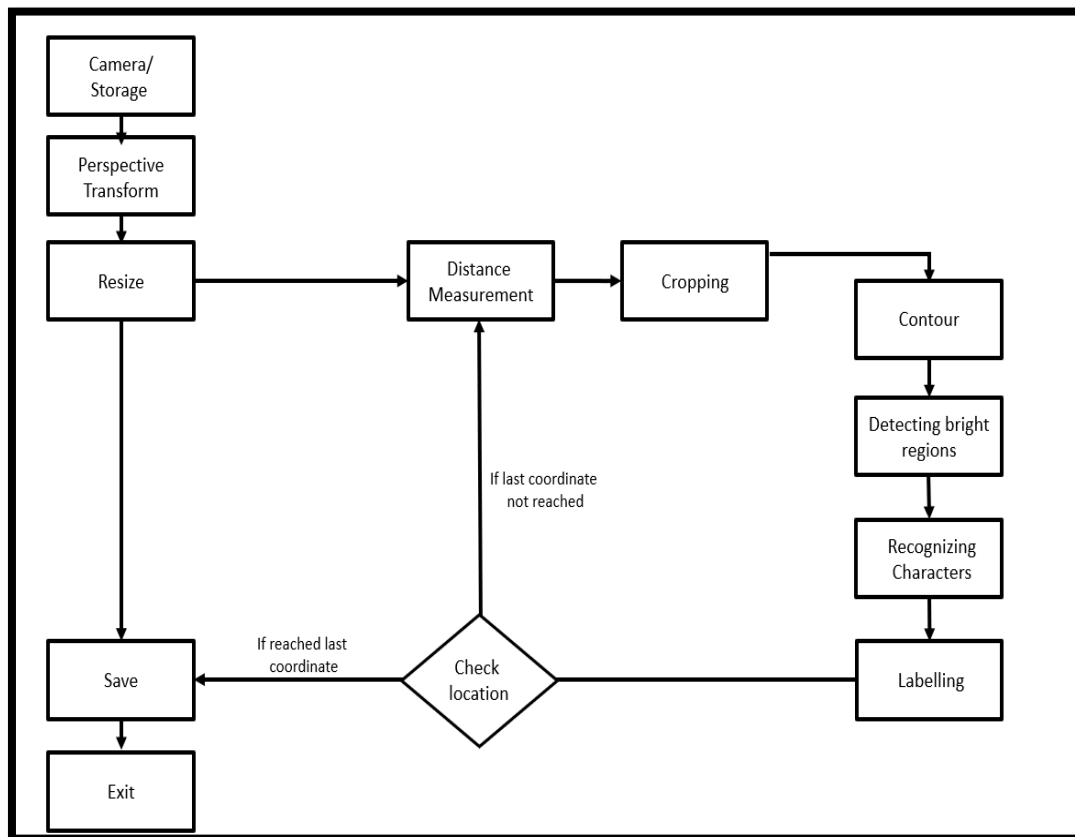


Fig 1.11: DRE Model Workflow

This method involves a simple DRE model (Detection, Recognition, and Execution) In the starting user adds the image from storage or captures it through a phone camera. The algorithm applies the perspective transformation

- Step 1: Detect edges.
- Step 2: Use the edges in the image to find the contour (outline) representing the piece of paper being scanned.
- Step 3: Apply a perspective transform to obtain the top-down view of the document.

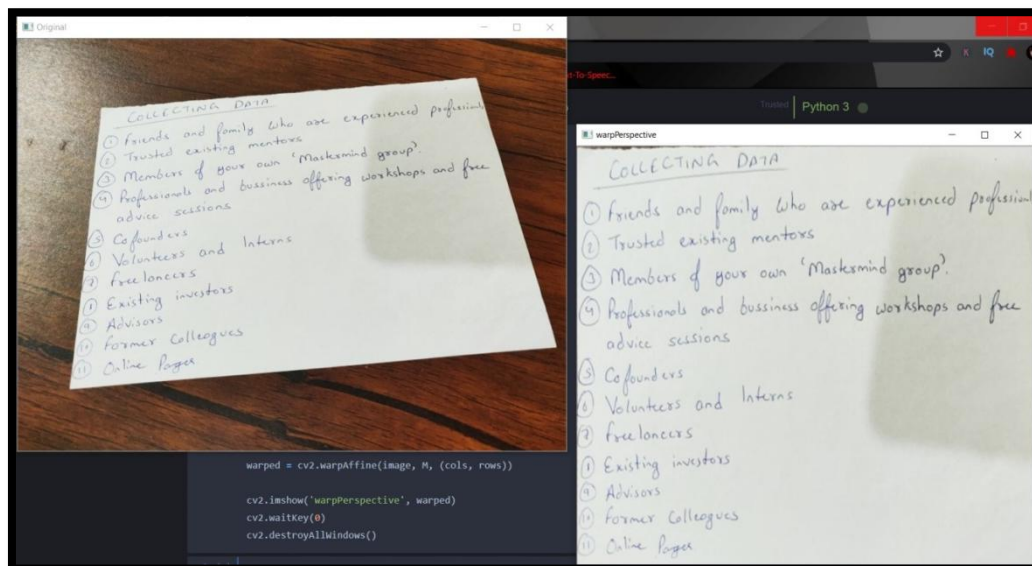


Fig 1.12 Perspective transformation of the image

The coordinates of the image required for perspective transform were added using our app through a simple slide and selection method.

### Resizing the image captured after perspective transformation

After resizing, the image enters the loop.

NOTE: Every braille dot was considered as a 3d object with all properties of a 3d object.

### DETECTION

Steps:

- Start by detecting dots on the paper for further processing.
- Import all required python packages.
- Load image available after resizing
- Convert image to grayscale
- Blur image using a Gaussian filter with a  $7 \times 7$  kernel.
- Once the image has been blurred, apply the Canny edge detector to detect edges in the image
- Morphological operations (dilation + erosion) are then performed to close any gaps in the edge map<sup>(4)</sup>

A call to `cv2.findContours` detects the outlines of the objects in the edge map while sorting our contours from left-to-right.

We then initialize a list of colors used to draw the distances along with the `refObj` variable, which will store our *bounding dot*, *centroid*, and *pixels-per-metric* value of the reference dot.

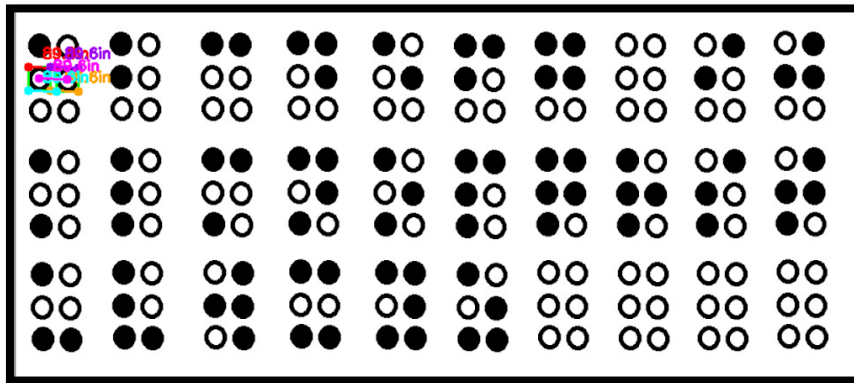


Fig 1.13: Calculating the distance of nearby dark braille dots/circles

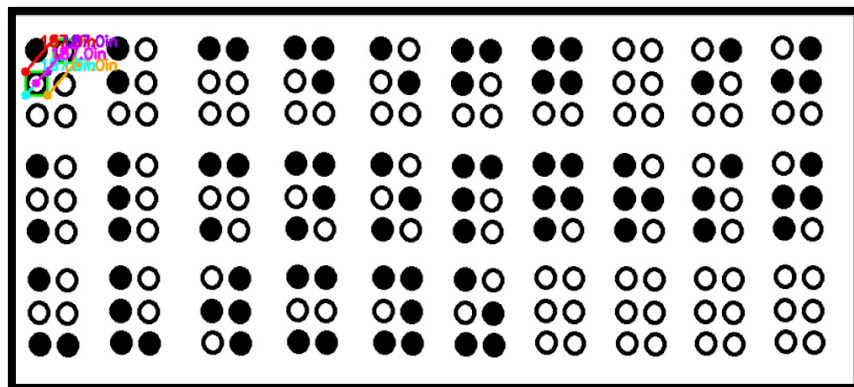


Fig 1.14: Calculating the distance of nearby dark braille dots/circles

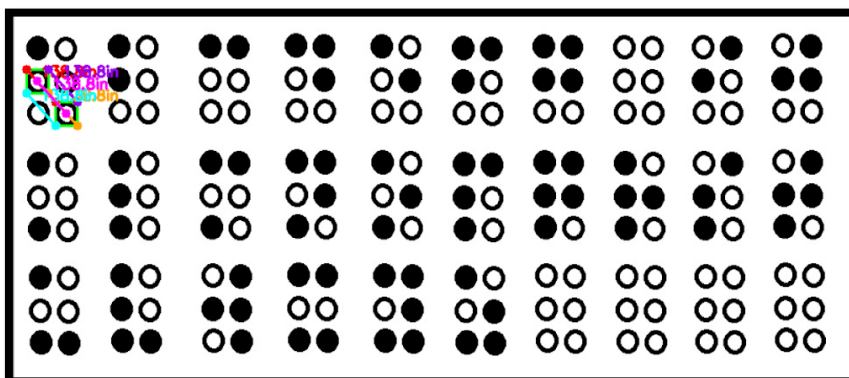


Fig 1.15: Calculating the distance of nearby dark braille dots/circles

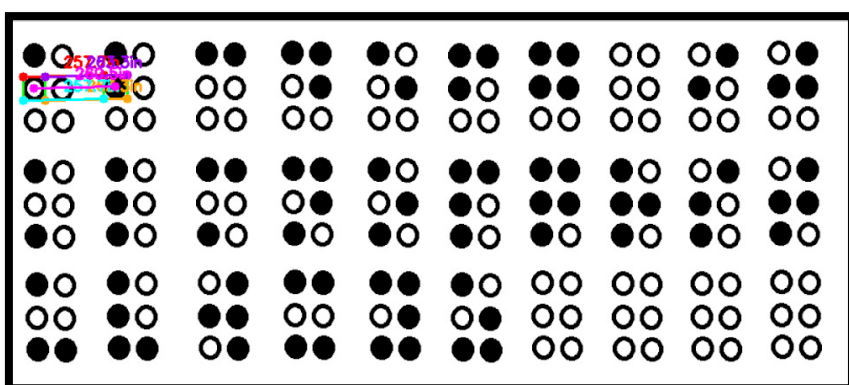


Fig 1.16: Calculating the distance of another braille character for determining the group of six dots

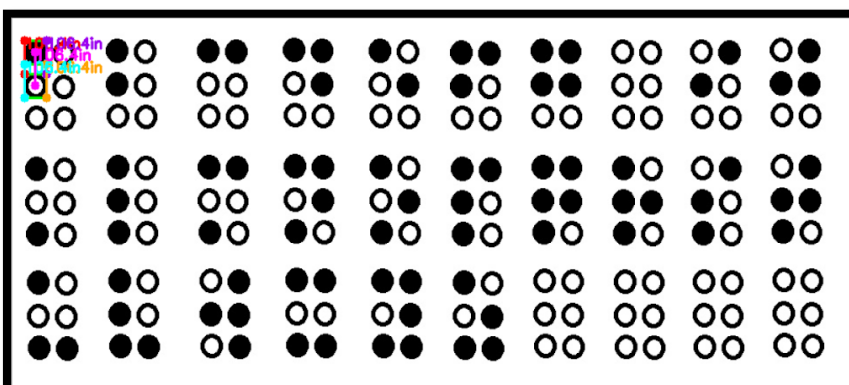


Fig 1.17: Calculating the distance of nearby dark braille dots/circles

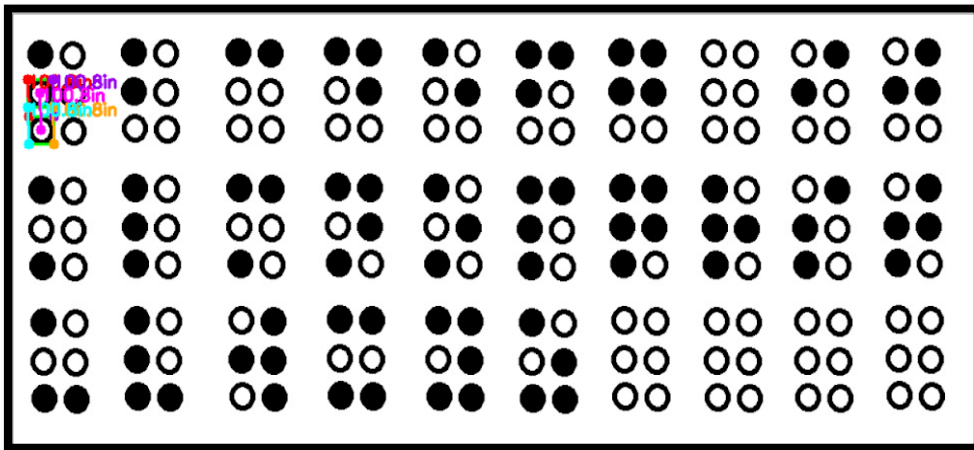


Fig 1.18: Calculating the distance of nearby dark braille dots/circles

After calculating the distance, Our algorithm:

- Start looping over each of the contours in the list.
- If the contour is not sufficiently large, we ignore it.
- Otherwise, the rotated bounding box of the current object (using cv2.boxPoints for OpenCV 3).
- A call to order points rearranges the bounding box (x, y)-coordinates in top-left, top-right, bottom-right, and bottom-left order, which as we'll see, is important when we go to compute the distance between object corners. Then we compute the center (x, y)-coordinates of the rotated bounding box by taking the average of the bounding box in both the x and y direction.
- Start by unpacking the (ordered) rotated bounding box coordinates and computing the midpoint between the top-left and bottom-left along with top-right and bottom-right points. From there, compute the Euclidean distance between the points, giving us our "pixels-per-metric", allowing us to determine how many pixels fit into –width inches.
- Euclidean distance between the reference location and the object location, followed by dividing the distance by the "pixels-per-metric", giving us the final distance in inches between the two objects. The computed distance is then drawn on our image

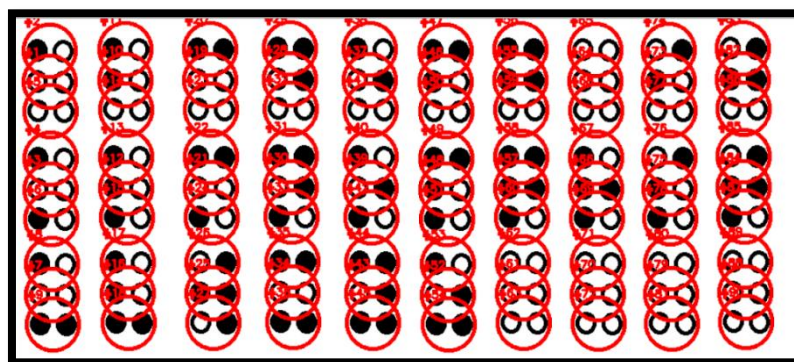


Fig 1.19: Grouped sets of braille character (6 dots)

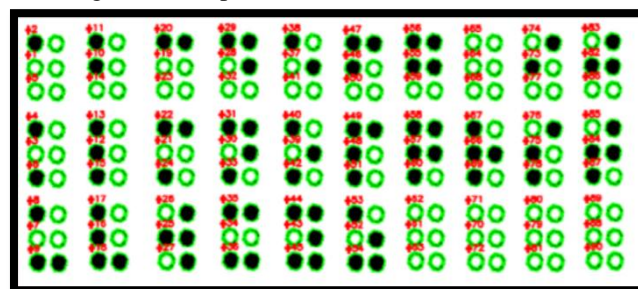


Fig 1.20: Grouped sets of braille character (6 dots)

All the circles with 200inch are considered to be a single braille character and cropped into the next loop for further processing.

For each of these contours, we will compute the minimum enclosing circles (6) which represent a single character in the braille system.

First, we detected the contours in the mask and then sorted them from left-to-right. Once our contours have been sorted, we loop over each cropped character individually with their location coordinates.

## RECOGNITION

**CROPPING:** The six dots closed to each other based on the distance between them are cropped.

Apply thresholding

This operation takes any pixel value  $p \geq 200$  and sets it to 255 (white). Pixel values  $< 200$  are set to 0 (black). We used binary inverse thresholding because we wanted our braille dark dots to be bright for better recognition. All our backgrounds and all unnecessary objects should be black.

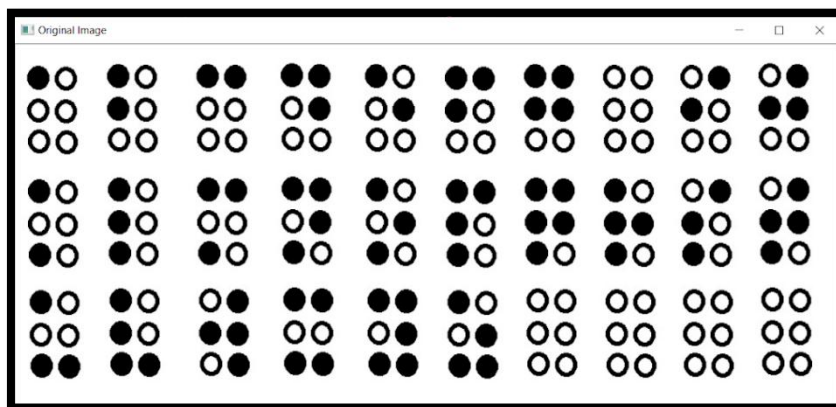


Fig 1.21: Original Computer-Generated braille character

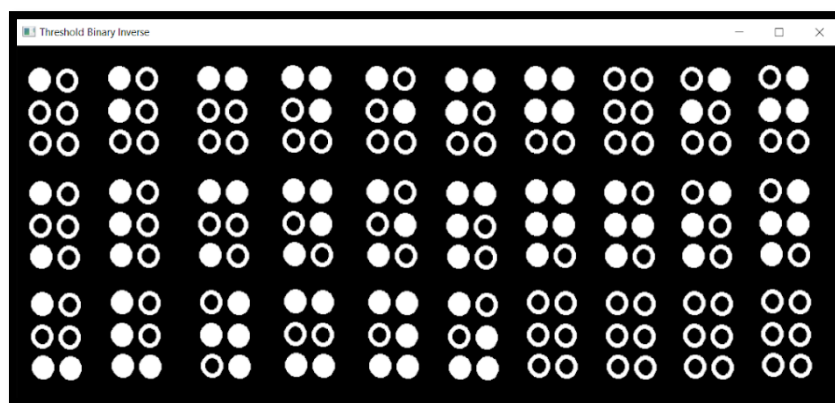


Fig 1.22: Binary inverted thresholding of the original image

However, there is a bit of noise in this image, so we cleaned it up by performing a series of erosions and dilations.

We then uniquely label the brighter regions and draw it on the image with no. label starting from top left to right bottom circle, which should be == 6

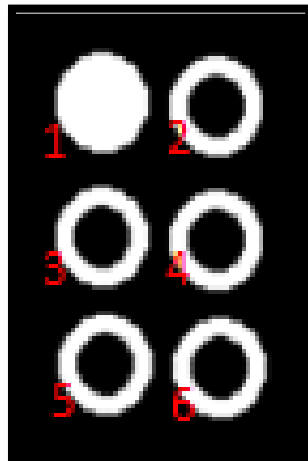


Fig 1.23 Labelled braille character box

### I – Creation of unfilled dots in spaces with no dots

- Using edge detection and MATLAB, calculate the area of each dot in the image.
- Remove the area values of the largest 10% and the smallest 10%. This is done to remove outliers.
- Take an average of all remaining areas
- Filter all dots out of the image whose average area is more than 10% above or below the average area value of each dot
- Using center points of all the filtered dots in the image replace all the dots with circles of average dot size
- A basic neural network with a limited database can be used to identify characters.

### II -Recognition

- Take the topmost left dot
- Search along the x-axis and y-axis to find the minimum distance between 2 dots
- Save this value as A
- Distance between dot points (1 and 2), (2 and 3), (4 and 5), (5 and 6), (2 and 5), (3 and 6), (1 and 4) is A
- Distance between points (1 and 5), (2 and 4), (3 and 5), (2 and 6) is  $A\sqrt{2}$
- Distance between points (1 and 6), (3 and 4) is  $A\sqrt{5}$

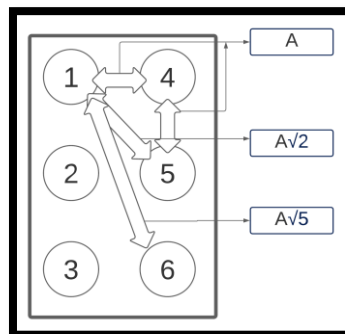


Fig 1.24

After grid separation, the algorithm implements a distance calculation algorithm.

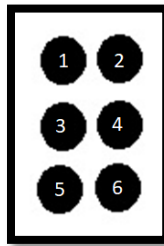


Fig 1.25: Labelling the no. of each dot.

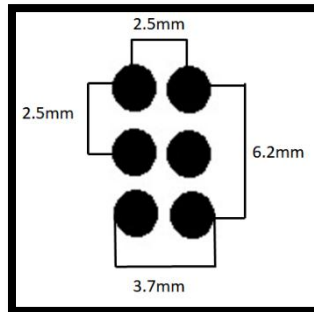


Fig 1.26: Showcasing standard distance between the braille dots.

Based on these standard distances our algorithm will recognize the braille character.  
Before which,

a. When dot 1 is taken as refObj, the distances between braille dots are taken as variables:

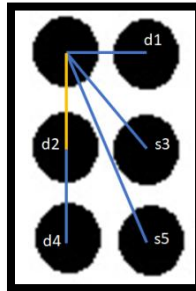


Fig 1.27: Variable Labelling with respect torefObj dot 1

- d1 for the distance between dots 1 and 2.
- d2 for the distance between dots 1 and 3.
- s3 for the diagonal distance between dots 1 and 4.
- d4 for the distance between dots 1 and 5.
- S5 for the diagonal distance between dots 1 and 6.

b. When dot 2 is taken asrefObj, the distances between braille dots are taken as variables:

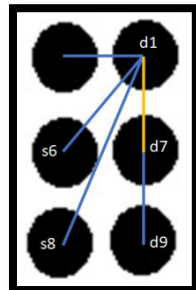


Fig 1.28: Variable Labelling with respect torefObj dot 2



- d1 for the distance between dots 2 and 1.
- s6 for the diagonal distance between dots 2 and 3.
- d7 for the distance between dots 2 and 4.
- s8 for the diagonal distance between dots 2 and 5.
- d9 for the distance between dots 2 and 6.

After the algorithm detects the distances between available dots it stores the available values in the above-mentioned variable, depending on the refObj.

Note: The variables for which values aren't available will be taken zero.

Then the above values are passed on to the formula:

$$D = |R + r + d(d1+d2+d4+d5+d7+d9) + s(s3+s5+s6+s8)|$$

where,

D is the total distance of all dots,

R is the radius of 1st dot and r is the distance of 2nd dot, though both will be the same

s is the diagonal distance.

d is the distance between two dots

In the case of no diagonal distance, the value of s will be taken zero.

In the case of diagonal distance, the value of d will be taken zero.

The recognized value of each variable should be around standard values:

$$R=r=0.5\text{mm}$$

When refObj is dot 1:

- d1 = 2.50 mm
- d2 = 2.50 mm
- s3 = 3.53 mm
- d4 = 5.00 mm
- s5 = 5.59 mm

When refObj is dot 2:

- d1 = 2.50 mm
- s6 = 3.53 mm
- d7 = 2.50 mm
- s8 = 5.59 mm
- d9 = 5.00 mm

Other median distances from nearby cells for setting the range of formula:

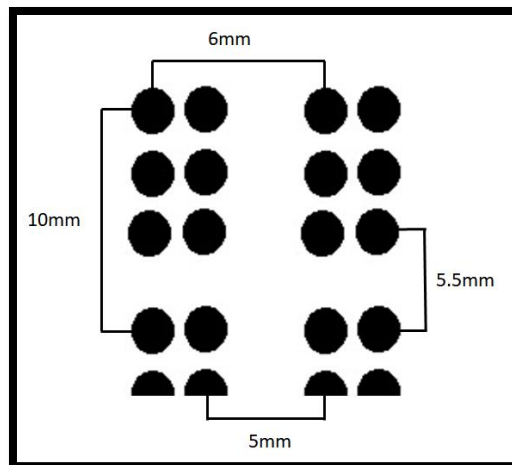



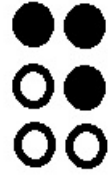
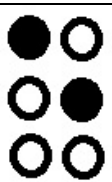



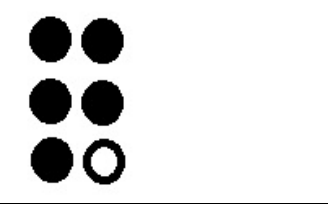
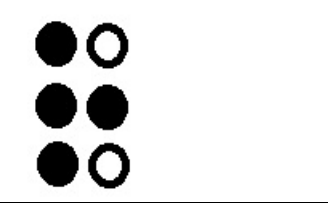
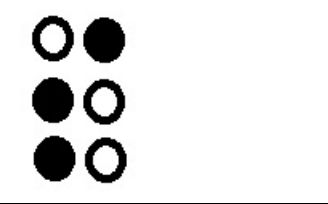
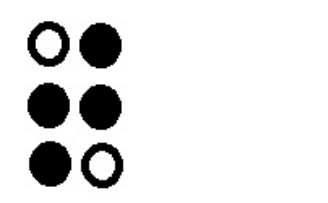
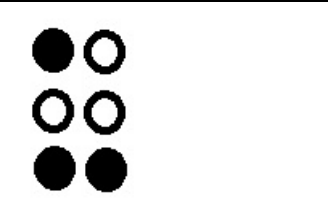
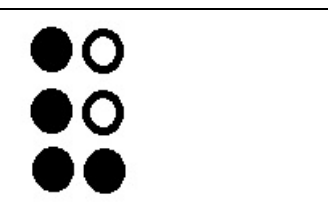
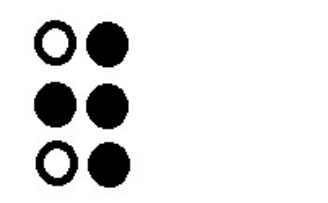
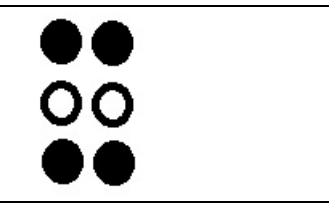




Fig 1.29 Distance between neighboring braille dots

An array is created containing distances of all dots at each range for all alphabets w.r.t refObj  
Values will be stored by the algorithm. Here we have shown example values.

Braille Character	English Alphabet	Values (in mm)		Total Distances
	A	d1 = 0 d2 = 0 s3 = 0 d4 = 0 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=0
	B	d1 = 0 d2 = 2.5 s3 = 0 d4 = 0 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=2.5
	C	d1 = 2.5 d2 = 0 s3 = 0 d4 = 0 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=2.5
	D	d1 = 2.5 d2 = 0 s3 = 3.53 d4 = 0 s5 = 0	s6 = 0 d7 = 2.5 s8 = 0 d9 = 0	D=8.53
	E	d1 = 0 d2 = 0 s3 = 3.53 d4 = 0 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=3.53
	F	d1 = 2.5 d2 = 2.5 s3 = 0 d4 = 0 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=5.00
	G	d1 = 2.5 d2 = 2.5 s3 = 3.53 d4 = 0 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=8.53
	H	d1 = 0 d2 = 2.5 s3 = 3.53 d4 = 0 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=6.03

	I	d1 = 0 d2 = 0 s3 = 0 d4 = 0 s5 = 0	s6 = 3.53 d7 = 0 s8 = 0 d9 = 0	D=3.53
	J	d1 = 0 d2 = 0 s3 = 0 d4 = 0 s5 = 0	s6 = 3.53 d7 = 2.5 s8 = 0 d9 = 0	D=6.01
	K	d1 = 0 d2 = 0 s3 = 0 d4 = 5 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=5.00
	L	d1 = 0 d2 = 2.5 s3 = 0 d4 = 5 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=7.50
	M	d1 = 2.5 d2 = 0 s3 = 0 d4 = 5 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=7.50
	N	d1 = 2.5 d2 = 0 s3 = 3.53 d4 = 5 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=11.01
	O	d1 = 0 d2 = 0 s3 = 3.53 d4 = 5 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=8.53
	P	d1 = 2.5 d2 = 2.5 s3 = 0 d4 = 5 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=10.00

	Q	d1 = 2.5 d2 = 2.5 s3 = 3.53 d4 = 5 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=13.23
	R	d1 = 0 d2 = 2.5 s3 = 3.53 d4 = 5 s5 = 0	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=11.03
	S	d1 = 0 d2 = 0 s3 = 0 d4 = 0 s5 = 0	s6 = 3.53 d7 = 0 s8 = 5.59 d9 = 0	D=10.12
	T	d1 = 0 d2 = 0 s3 = 0 d4 = 0 s5 = 0	s6 = 3.53 d7 = 2.5 s8 = 5.59 d9 = 0	D=11.62
	U	d1 = 0 d2 = 0 s3 = 0 d4 = 5 s5 = 5.59	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=10.59
	V	d1 = 0 d2 = 2.5 s3 = 0 d4 = 5 s5 = 5.59	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=13.09
	W	d1 = 0 d2 = 0 s3 = 0 d4 = 0 s5 = 0	s6 = 3.53 d7 = 2.5 s8 = 0 d9 = 5	D=11.03
	X	d1 = 2.5 d2 = 0 s3 = 0 d4 = 5 s5 = 5.59	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=13.09

	Y	d1 = 2.5 d2 = 0 s3 = 3.53 d4 = 5 s5 = 5.59	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=16.62
	Z	d1 = 0 d2 = 0 s3 = 3.53 d4 = 5 s5 = 5.59	s6 = 0 d7 = 0 s8 = 0 d9 = 0	D=14.12

Out of available darkened dots, the algorithm checks the distance of recognized dots in the grid from the table. Depending on the concurrent distance it checks that all values fall in the set range and labels the alphabet.

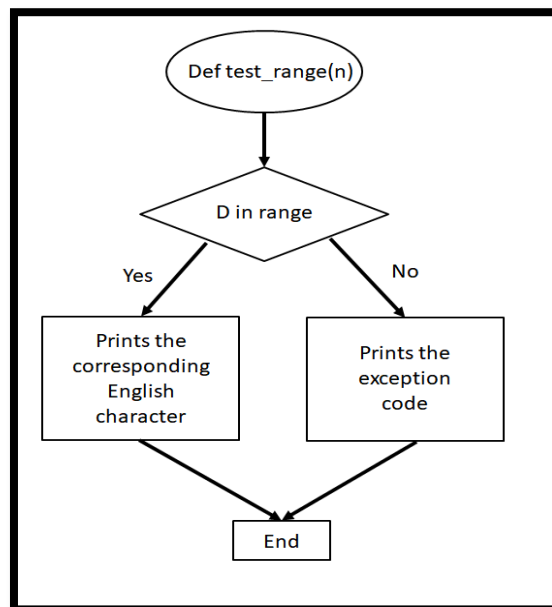


Fig 1.30: Algorithmic Flowchart for checking given total distance is calculated for dots in one braille character set only.

**Inference**

With this, the given character is recognized, and the algorithm will exit the execution loop. It will enter the detection loop again, calculate the distance then goes into the recognition loop, after cropping, again into the execution loop.

This continues until the last character is reached.

After this, a full document with all labelled characters will be shown to the user. Users can scan another document or simply save and exit the application.

Using 3 loop methods called DER we were able to detect braille characters.

**IV. REFERENCES**

- [1]. R. Fisher, S. Perkins, A. Walker, E. Wolfart. (2003). Gaussian Smoothing [Online]. Available at: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm> (Accessed: July 2020).
- [2]. Team Mathworks. (2020). Morphological Operations (Dilate, erode, reconstruct) [Online]. Available at: <https://in.mathworks.com/help/images/morphological-filtering.html#:~:text=Types%20of%20Morphological%20Operations,erosion%20for%20more%20specialized%20operations> (Accessed: July 2020).
- [3]. Alexander Mordvintsev, Abid K. (2013). Contour Properties. [Online]. Available at: [https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_imgproc/py\\_contours/py\\_contour\\_properties/py\\_contour\\_properties.html?highlight=grayscale](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contour_properties/py_contour_properties.html?highlight=grayscale) (Accessed: June 2020).
- [4]. Diane P. Wormsley. (2008). Braille is NOT a Language. [Online]. Available at: <http://www.brailleauthority.org/notalanguage/braille-is-not-a-language.html> (Accessed: July 2020). <sup>[1]</sup>

- [5]. American Foundation for the Blind. (2020). What Is Braille? [Online]. Available at: <https://www.afb.org/blindness-and-low-vision/braille/what-braille> (Accessed: May 2020).<sup>[2]</sup>
- [6]. Braille Authority of North America. (2010). Size and Spacing of Braille Characters. [Online]. Available at: <http://www.brailleauthority.org/sizespacingofbraille/index.html> (Accessed: July 2020).<sup>[3]</sup>
- [7]. Sharon Duffy. (1990). Grade Three Braille Compared To Other Abbreviated Forms of Braille. [Online]. Available at: <https://www.nfb.org/sites/www.nfb.org/files/images/nfb/publications/fr/fr9/issue2/f090212.html> (Accessed: xx 2020).<sup>[4]</sup>
- [8]. Renu Khandelwal. (2019). SSD: Single Shot Detector for object detection using MultiBox. [Online]. Available at: <https://towardsdatascience.com/ssd-single-shot-detector-for-object-detection-using-multibox-1818603644ca> (Accessed: May 2020).<sup>[5]</sup>
- [9]. Alexander Mordvintsev, Abid K. Revision. (2013). Canny Edge Detection. [Online]. Available at: [https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_imgproc/py\\_canny/py\\_canny.html](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html) (Accessed: May 2020)<sup>[6]</sup>
- [10]. Dimitri van Heesch. (2020). Contours Hierarchy and Image. [Online]. Available at: [https://docs.opencv.org/master/d9/d8b/tutorial\\_py\\_contours\\_hierarchy.html](https://docs.opencv.org/master/d9/d8b/tutorial_py_contours_hierarchy.html) (Accessed: June 2020)<sup>[7]</sup>
- [11]. PharmaBraille. (2020). The Braille Alphabet. [Online]. Available at: <https://www.pharmabraille.com/pharmaceutical-braille/the-braille-alphabet/> (Accessed: xx 2020)<sup>[8]</sup>
- [12]. T. D. S. H. Perera, W. K. I. L. Wanniarachchi. Optical Braille Translator For Sinhala Braille System: Paper Communication Tool Between Vision Impaired And Sighted Persons. *The International Journal of Multimedia & Its Applications (IJMA)*, vol.10, no.1/2/3. Available from: [June 2018].
- [13]. Aisha Mousa, Hazem Hiary, Raja Alomari, LoaiAlnemer. Smart Braille System Recognizer. *IJCSI International Journal of Computer Science Issues*, Vol. 10, Issue 6, No 1. Available from: [November 2013]