International Journal of Engineering Trends and Technology
ISSN: 2231-5381/ https://doi.org/10.14445/22315381/IJETT-V73I3P130

Volume 73 Issue 3, 436-447, March 2025
© 2025 Seventh Sense Research Group®

Original Article

Southern Indian Language Braille Image Conversion Systems

G. Gayathri Devi

Department of Computer Science, Shrimathi Devkunvar Nanalal Bhatt Vaishnav College for Women, Chennai, India.

Corresponding Author : mail2gg@yahoo.co.in

Received: 21 August 2024 Revised: 14 November 2024 Accepted: 25 February 2025 Published: 28 March 2025

Abstract - *The Southern Indian Language Braille Image Conversion System (SILBCS) is designed to enhance accessibility for visually impaired individuals by converting Braille script into readable text in Southern Indian languages. By focusing on the recognition and interpretation of Braille Image characters specific to regional languages such as Tamil, Telugu, Kannada, and Malayalam, SILBCS offers a tailored solution to address the linguistic diversity prevalent in the area. This system inputs Braille image documents and generates corresponding text files in languages such as Tamil, Telugu, Kannada, and Malayalam. The converted text can be further utilized with speech synthesizers to enable audio output. Through experimentation with datasets containing Braille documents in these languages, the proposed method demonstrated promising performance in accurately converting Braille images into their respective languages, thus facilitating improved access to information and literature for the visually impaired community in Southern India.*

Keywords - *Braille conversion, Kannada Braille, Malayalam Braille, Southern Indian Language, Tamil Braille, Telugu Braille.*

1. Introduction

People with vision problems use Braille, which is credited to Louis Braille and has been widely discussed by (Barry 1981) [1] and (Jan Mennens et al. 1993, 1994) [2-4]. Braille is a tactile writing system. Those with low vision or blindness can read thanks to its raised dots, which are detectable by touch. As seen in Figure 1, the system is divided into Braille cells, each of which has a configuration of six raised dots grouped in two rows of three. Six dots can be used to create sixty-four (2^6) different combinations, which can be used to represent numerals, punctuation, alphabet letters, and even whole sentences in a single cell.

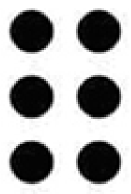


Fig. 1 Braille cell

The Dravidian languages, primarily spoken in Tamil Nadu, Kerala, Telangana, Andhra Pradesh, and Karnataka, extend their influence to regions such as Bangladesh, Bhutan, Mauritius, Sri Lanka, certain parts of Pakistan, Burma, southern Afghanistan, Nepal, Malaysia, Africa, Indonesia, and Singapore. Tamil, Kannada, Telugu, and Malayalam boast the highest number of speakers among these languages. Braille serves as an essential tool enabling individuals worldwide to access information in their native languages, thus facilitating the universal spread of knowledge.

Known as Indian Braille or Bharati Braille [5, 6], it is widely used as a Braille alphabet system for writing Indian languages. Initially, eleven distinct Braille scripts were utilized, each tailored to different languages in various parts of India. However, Bharati Braille has become a standardized script nationwide and has been adopted by countries such as Bangladesh, Sri Lanka, and Nepal. Figures 2, 3, 4, and 5 visually depict the Braille alphabets for Tamil, Telugu, Malayalam, and Kannada, respectively.

2. Existing System

The (Andrean et al. 2023) [7] research focused on the preprocessing techniques to convert text images to braille Unicode. The research uses preprocessing techniques such as binarization, denoising, and skew correction, and then Optical Character Recognition (OCR) is applied. This paper came up with the results of the preprocessing methods and compared those results to results from raw images. The work emphasized the importance of preprocessing procedures in enhancing text images for braille translation. The proposed research (Sana Shokat et al., 2022) [8] adopted Support Vector (SVM), Decision Trees (DT), and K-Nearest Neighbor (KNN) algorithms, combined with Reconstruction Independent

Component Analysis (RICA) and Principal Component Analysis (PCA) based feature extraction methods for converting Braille image to English.

The work found that the RICA-based feature extraction method worked well compared to PCA, the Random Forest (RF) algorithm, and Sequential methods. Evaluation metrics included True Positive Rate (TPR), True Negative Rate (TNR), Positive Predictive Value (PPV), Negative Predictive Value (NPV), False Positive Rate (FPR), Total Accuracy, Area Under the Receiver Operating Curve (AUC), and F1-Score. The robustness of the proposed method was confirmed by conducting statistical tests.

The paper (F. S. Apu et al., 2021) [9] proposed a novel Braille translating device capable of converting text and voice commands into Braille characters for visually challenged people. The device features a single refreshable Braille cell to display the Braille signs. The device is operable through smartphones and computers using Bluetooth and USB connections, allowing educators to connect and manage up to 30 Braille translating devices simultaneously to teach a class of 30 students.

The aim of the research (Felix et al., 2020) [10] was to develop a software system capable of converting Braille code into multilingual text and voice outputs using Machine Learning (ML) and Artificial Intelligence (AI) algorithms. The research faced challenges in the variability in Braille writing styles and solved those problems by using ML and AI models that can adapt to these differences. The research found a comprehensive solution that efficiently enhanced the accessibility of written content for the visually challenged and enabled them to benefit from both text and voice outputs in multiple languages.

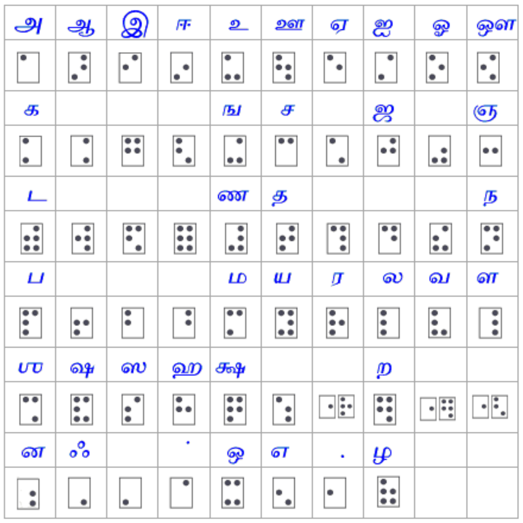


Fig. 2 Tamil braille alphabets sheet

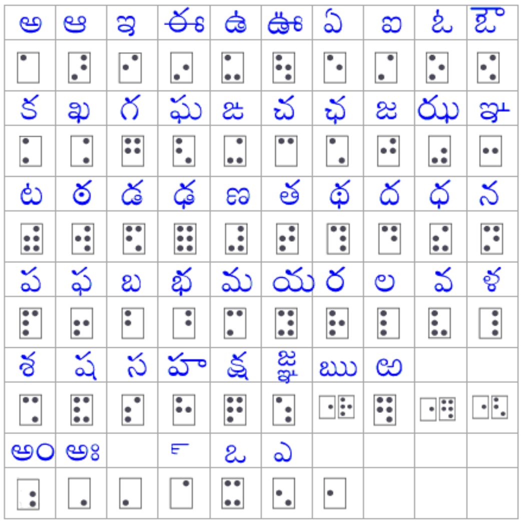


Fig. 3 Telugu braille alphabets sheet

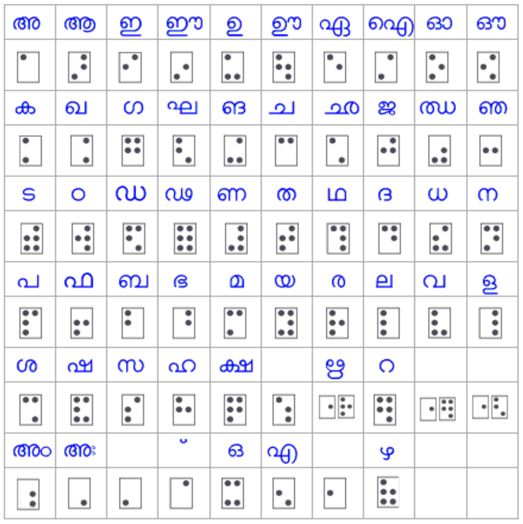


Fig. 4 Malayalam braille alphabets sheet

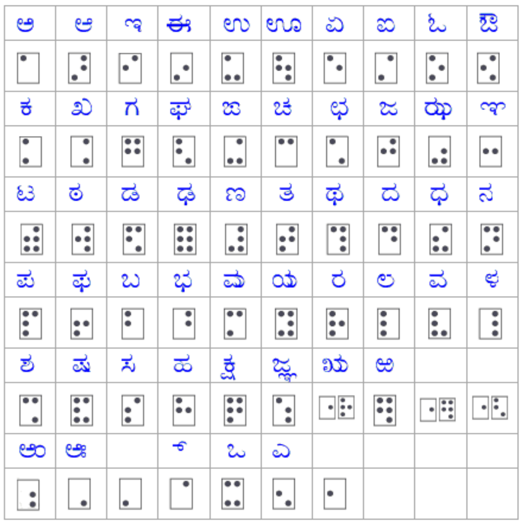


Fig. 5 Kannada braille alphabets sheet

This work (Purvang Patel et al., 2021) [11] addresses the real-time teaching needs of visually challenged students to learn alongside their peer group. The system used a webcam, Tesseract OCR, an Arduino board, and solenoids as the output mechanism for blind students to feel the Braille characters. The algorithm extracts the text using Tesseract OCR and converts the recognized text into Braille. The recognized Braille is created using an Arduino board, providing a real-time learning experience for visually impaired students.

The work (Ghazanfar et al., 2023) [12] introduced an innovative IoT-based system to translate Arabic and English Braille into audio. The device captures Braille images with an embedded camera and employs a transfer learning-based Convolutional Neural Network (CNN) for recognition. This system helps visually impaired individuals and other people to read Braille.

The research (Shokat et al.,2020) [13] reviewed various user input schemes such as touch-screen mechanisms like Braille Touch, Braille Enter, and Edge Braille, highlighting their reliance on location-specific inputs, which could be challenging for the user to convert Braille to natural language. The study evaluated the efficiency, accuracy, and usability techniques to minimize user burden.

The article (Changjian Li et al., 2021) [14] focused on implementing a character-based braille translator using ResNet models (ResNet-18, ResNet-34, ResNet-50) and a word-based braille detector using Adaptive Bezier-Curve Network (ABCNet). The research paper presented a comparative evaluation of ABCNet’s performance. The approach leverages deep learning techniques to enhance braille recognition and detection, aiming to improve accessibility and usability for visually impaired individuals.

The research article (Englebretson R (2023) [15] presented an approach for recognizing Braille as a distinct and valuable writing system deserving of scholarly attention. The article provides a historical overview of Braille’s development, focusing on its unique formal characteristics as a writing system to familiarize sighted print readers and encourage further exploration. It critiques biases that prioritize print-centric assumptions, highlighting the negative impact on braille users and research directions.

The paper (Jennis Oliviya et al., 2023) [16] introduced a method to bridge communication gaps between sighted and visually impaired individuals using Tamil braille. The research proposed a transfer learning approach employing the VGG16 network to convert Tamil braille images into text with a high accuracy of 95.65%. The research (Ian Herrara et al., 2024) [17] describes the development of a Braille printer that converts voice input into Braille, utilizing an old ink printer as a base. The innovation aims to improve communication and education for deafblind individuals by enabling real-time voice-to-Braille translation. The process involves designing a prototype, modifying the printer mechanism, and implementing a control system.

By repurposing materials and leveraging existing printer infrastructure, the approach not only enhances accessibility but also promotes cost-effectiveness and sustainability. The work (Urooj Beg 2017) [18] converted scanned Hindi documents into Braille using image processing. The method generates a Hindi database through segmentation, matches letters with Principal Component Analysis (PCA), and converts them into Braille. Text documents are successfully translated, marking progress in Braille communication research.

The proposed research (Ganga Gudi et al., 2023) [19] highlighted the challenges visually impaired individuals face in accessing information compared to sighted people, stressing the importance of developing a system for converting natural language text into Braille. The paper proposed strategic mapping solutions using Natural Language Processing (NLP) techniques to optimize multilingual text translation into Braille characters.

The work (B. Meneses-Claudio et al. 2020) [20] addressed the educational problem faced by visually challenged persons in Peru by developing a Braille translation system. This system captures Braille images and translates them into Peruvian vocabulary using image processing techniques. By identifying and interpreting the Braille points, the approach seeks to improve understanding and accessibility for both students and educators involved in teaching Braille.

3. Proposed Methodology

The current research presents a conversion system from southern Indian Braille languages to their respective spoken languages. The datasets primarily comprised Braille documents in Southern Indian languages, including Tamil, Telugu, Kannada, and Malayalam. These were sourced from educational institutions catering to visually impaired students, Braille libraries housing regional literature, and custom-created scanned samples of Braille texts using high-resolution scanners. The collected datasets included textbooks, literary works, and worksheets, ensuring a mix of content types.

The research encompasses the following phases:

- Braille Image Preprocessing,
- Row-Column Segmentation,
- Conversion of Braille Cell to a number string
- Creation of Mapping Table
- Conversion of the number string to its Unicode representation
- Conversion of the Unicode representation to editable text file.

The workflow of the proposed system is shown in Figure 6.

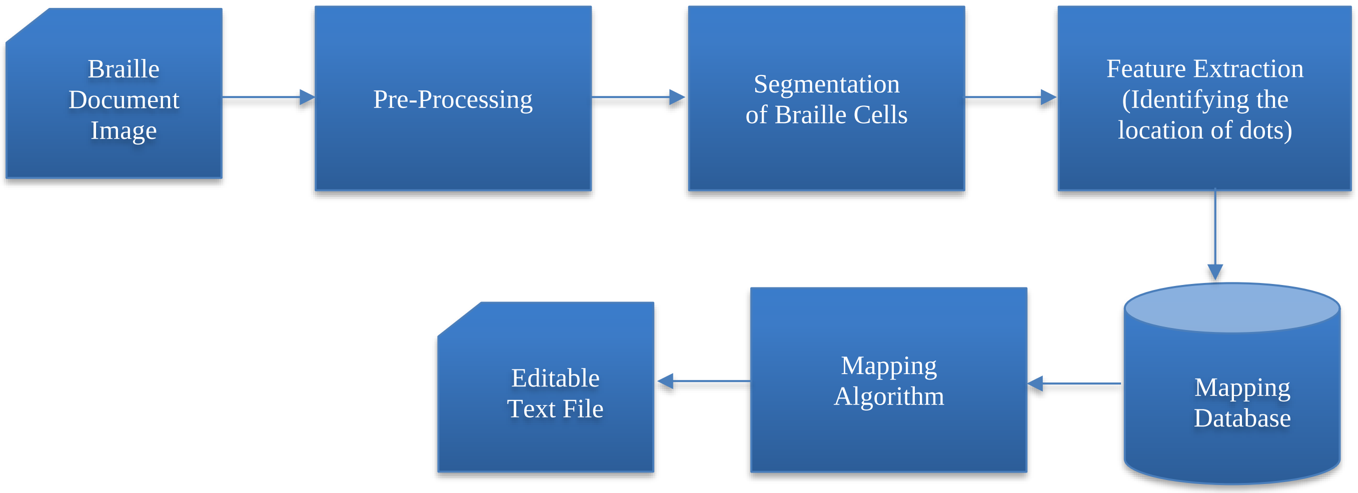


Fig. 6 Workflow of proposed system

3.1. Braille Image Preprocessing

Preprocess the Braille image to enhance its quality and prepare it for subsequent processing stages, such as segmentation and character recognition. Preprocessing a Braille image involves the following steps. The first step is to load the image from the file system into memory. The image is then converted to grayscale to simplify processing and reduce computational load.

Gaussian blurring is then applied to the grayscale image to smooth it out, reducing noise and eliminating small details that may interfere with subsequent processing steps. Next, a thresholding technique is employed to segment the image into binary form, effectively separating the foreground (containing Braille dots) from the background.

After thresholding, morphological operations are applied. Dilation is performed to connect nearby Braille dots and ensure they form coherent structures, while erosion is utilized to remove any remaining small noise and separate merged dots. Contour detection is then employed to identify individual Braille dots or cells within the processed image. Following contour detection, contours that are too small or too large to be considered Braille dots are filtered out.

Bounding boxes are then extracted around the remaining contours to isolate individual Braille cells. Subsequently, the image is cropped using these bounding boxes to focus on each Braille cell individually. Finally, the cropped Braille cells are resized to a standard size for uniformity in further processing. The preprocessed Braille cell images are then saved for further analysis or processing in subsequent steps.

3.1.1. Braille Image Preprocessing Algorithm

Let I be the input Braille image.

1. Read the Image: I
2. *Convert to Grayscale:* I_{gray} =cvtColor(I,COLOR_BGR2GRAY)
3. Apply Gaussian Blurring: $I_{blurred}$=GaussianBlur(I_{gray} ,(5,5),0)
4. Thresholding: I_{thresh}=threshold($I_{blurred}$,0,255, THRESH_BINARY_INV + THRESH_OTSU)
5. Morphological Operations:

a. Dilation: $I_{dilation}$=dilate (I_{thresh},kernel,iterations=1)

b. Erosion: $I_{erosion}$=erode($I_{dilation}$,kernel,iterations=1)

6. Contour Detection: contours _=findContours(*Ierosion* , RETR_EXTERNAL, CHAIN_APPROX_SIMPLE)
7. Filter Contours: filtered_contours={cnt∣cntif$_{contour}$Area(cnt)>50and $_{contour}$Area(cnt)<1000}
8. Bounding Box Extraction and Image Cropping: preprocessed_images=[] for contour in filtered_contours:

- (x,y,w,h)=boundingRect(contour)
- braille_cell=I[y:y+h,x:x+w]
- preprocessed_images.append(braille_cell)

9. The output is the preprocessed image

A Tamil Braille Script image (Figure 7) is taken as Input, and after (Preprocess), Step 1 of the image is preprocessed and converted to a black and white image (Figure 8).

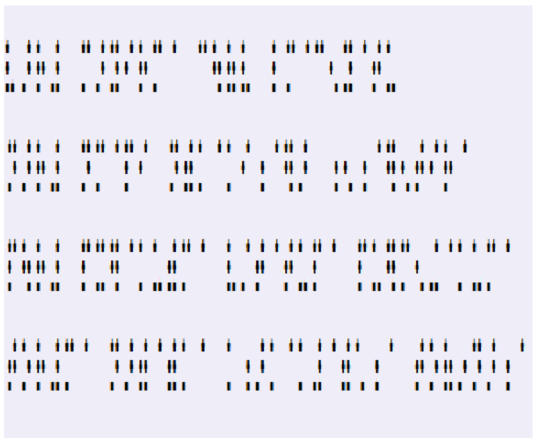


Fig. 7 Input - Tamil braille image

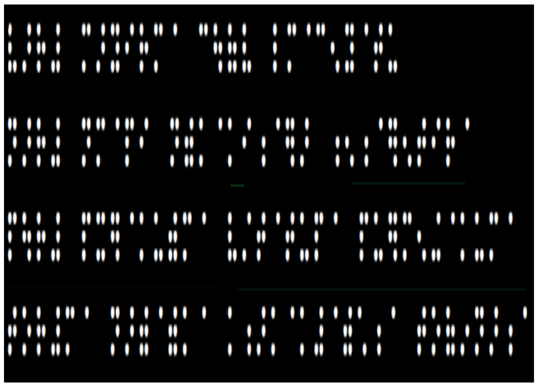


Fig. 8 Preprocessed -Tamil braille image

3.2. Row-Column Segmentation

Row-column segmentation of a preprocessed Braille document involves identifying and separating the individual rows and columns of Braille cells in the document. The procedure begins by identifying the rows of Braille cells. This is typically achieved by analyzing the vertical distribution of Braille cells within the document. Each row is delineated by the spaces between consecutive rows of cells. Once the rows are identified, the document is segmented horizontally into distinct rows. Following row segmentation, the individual columns within each row are identified. This is accomplished by analyzing the horizontal distribution of Braille cells within each row. Each column is separated by the spaces between consecutive columns of cells. The document is then segmented vertically into separate columns within each row. The resulting segmented document consists of individual cells organized in rows and columns, ready for further processing or analysis.

3.2.1. Braille Image Row-Column Segmentation Algorithm

Let D be the preprocessed Braille document.
Identify Rows:
Call RowSegmentFunc()

Let R represent the set of rows identified within D

- Each row r_i is defined by its starting and ending coordinates: (x_{start},y_{start},x_{end},y_{end}
- Segment Rows:

The segmented row SR_i is obtained by slicing D horizontally using the coordinates (x_{start},y_{start},x_{end},y_{end}).

Identify Columns Within Each Row:
Call ColumnSegmentFunc()

- Let C_n represent the set of columns identified within row r_i.
- Each column c_j within row r_i is defined by its starting and ending coordinates: (x_{start},y_{start},x_{end},y_{end}).

Segment Columns within Each Row:

- height(I) represents the height of the image I,
- count(y) denotes the number of non-zero pixels in row y,
- The segmented column $SC_{ri,j}$ is obtained by slicing SR_i horizontally using the coordinates x_{start},y_{start},x_{end},y_{end}).

Output

The resulting segmented document SD consists of individual Braille cells organized in rows and columns:
SD = {SR_1, SR_2,...,SR_n}

Where SRi represents a segmented row containing segmented columns:
SRi = {SC_{i1}, SC_{i2},...,$SC_{i,m}$}

RowSegmentFunc() - An Algorithm to Identify Rows in a Braille Image

The algorithm for identifying rows in a Braille image, represented, operates as follows: First, the preprocessed Braille image, denoted as D. An empty list called rows, is set to hold identified rows. The algorithm then iterates over each row in the image, computing the count of non-zero pixels in the row. If this count exceeds a predetermined threshold τ, the algorithm identifies the row as part of a Braille row segment.

If the start of the current row segment is not yet defined, it sets the start_row coordinate to the current row index. It updates the end_row coordinate with each subsequent row. If a row segment is completed, the algorithm appends the tuple (start_row, end_row) to the rows list and resets the start_row and end_row coordinates. After looping through all rows, the algorithm checks if the last row segment is still open and appends it to the rows list if needed. Finally, the algorithm outputs the list of identified rows.

```
RowSegemntFunc()
Let D be the preprocessed Braille image.
1. Set rows [] as an empty list.
2. Scan Over Images:
   • For each row y from 0 to height(D)−1:
   • Compute the number of non-zero pixels in the row, and count(y).
   • If count(y) is greater than or equal to a threshold τ:
   • If the start of the current row segment start_row is not set:
   • Set start_rowstart_row to y.
   • Update the end of the current row segment end_row to y.
     Else if
   • start_rowstart_row is set:
     Append the tuple (start_row,end_row) to rows R.
     Reset start_row_row and end_row to −1.
3. Return the list rows containing the identified rows R
```

In mathematical notation:

Rows = {(start_row,end_row) | start_row, end_row∈N,start_row≤end_row}
where:

- τ is the threshold for considering a row as part of a Braille row segment,

- start_rowstart_row and end_rowend_row are the start and end coordinates of a row segment, initialized to −1 at the beginning

Braille image is segmented into rows after the row segmentation algorithm is shown in Figure 9.

ColumnSegmentFunc() - An Algorithm to Identify Columns in a Braille Row Image

The algorithm to identify columns for segmentation in a Braille row image. The algorithm for identifying columns in a preprocessed Braille row image, denoted as R, follows a structured process. It begins by initializing an empty list named columns to hold the identified columns. Then, it iterates through each column in the image, calculating the count of non-zero pixels. If this count surpasses or equals a predefined threshold τ, the column is recognized as part of a Braille column segment. If the start of the current column segment hasn't been set, the start_column coordinate is established as the current column index. As it progresses through the columns, it updates the end_column coordinate accordingly. Upon completing a column segment, the tuple (start_column, end_column) is appended to the columns list and resets the start_column and end_column coordinates. After examining all columns, the algorithm verifies if the last column segment is still open and includes it in the columns list if necessary. Finally, it outputs the list columns containing the identified columns.

```
ColSegmentFunc()
Let R be the preprocessed Braille row image.
1.  Set columns as an empty list.
2.  Loop Over Columns:
•   For each column x from 0 to width(R)−1:
•   Compute the number of non-zero pixels in the column, count(x).
•   If count(x) is greater than or equal to a threshold τ:
•   If the start of the current column segment start_column is not set:
      Set start_column to x.
      Update the end of the current column segment and set end_column to x.
•   Else if
start_column is set
3.  Append the tuple (start_column1, end_column1) (start_column1, end_column1) to columns.
```

Output: Return the list of columns containing the identified columns.

In mathematical notation:

columns={(start_column,end_column) ∣ start_column,end_column∈N,start_column≤end_column}

where:

- width(R) represents the width of the image R,
- count(x) denotes the number of non-zero pixels in column x,
- τ is the threshold for considering a column as part of a Braille column segment,
- start_columnstart_column and end_columnend_column are the start and end coordinates of a column segment, initialized to −1 at the beginning. Adjust the threshold value according to the specific characteristics of your Braille row images. The Braille Row image is segmented into columns after the Column segmentation algorithm is shown in Figure 10.

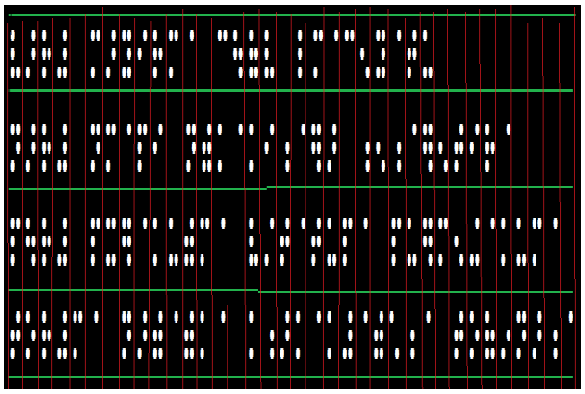


Fig. 9 Row projection-Tamil braille image

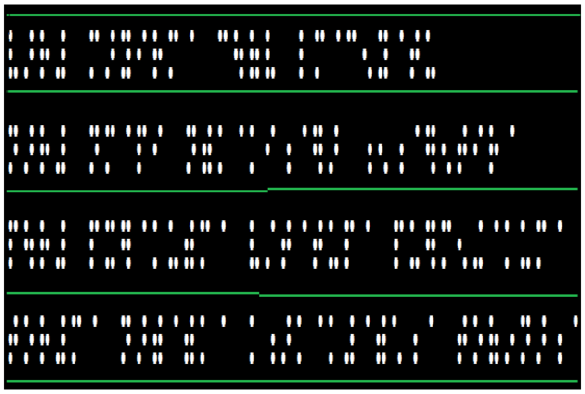


Fig. 10 Column projection-Tamil braille image

3.3. Conversion of Braille Cell to a Number String

To convert a Braille image pattern containing 6 dots into a numerical sequence, begin by identifying the positions of the dots within the pattern. Assign numerical values to each dot position, with the top left dot being labeled as 1, the middle left as 2, the bottom left as 3, the top right as 4, the middle right as 5, and the bottom right as 6.

The algorithm involves preprocessing the image to obtain a binary representation, detecting contours representing dots, filtering the contours, calculating centroids, and classifying dot positions. Once the dot positions are determined, map these positions to a numerical sequence. This sequence represents the numerical order of the dots within the Braille pattern.

Finally, the resulting numerical sequence is output, which provides a structured representation of the Braille image pattern in a numerical format. The algorithm is to convert a Braille image pattern consisting of 6 dots to a numerical sequence.

3.3.1. Dot Detection and Representation Algorithm

Input: Braille Character image pattern P consisting of a maximum of 6 dots.

1. Determine Dot Positions: Identify the positions of the dots within the Braille image pattern. Assign a numerical value to each dot position as follows:

Dot 1: Top left as 1
Dot 2: Middle left as 2
Dot 3: Bottom left as 3
Dot 4: Top right as 4
Dot 5: Middle right as 5
Dot 6: Bottom right as 6

2. Use contour detection algorithms to identify individual dots within the binary image:

dot_contours=detect_contours(P_{binary})

a. Filter the contours based on area and aspect ratio: filtered_contours=filter_contours(dot_contours)
b. For each remaining contour c_i in filtered_contours:
- Calculate the centroid$_i$ to determine the approximate position of the dot: centroid$_i$=calculate_centroid(c_i)
- Classify the dot based on its relative position within the grid: dot_position$_i$=classify_dot_position(centroid$_i$)

3. Map Dot Pattern to Numerical Sequence: Use the positions of the dots to map the Braille image pattern to a numerical sequence.

Output: Return the numerical sequence representing the Braille image pattern, i.e., the positions of the dots identified in the Braille image pattern: dot_positions = [dot_position$_1$, dot_position$_2$, ..., dot_position$_n$]

This algorithm takes a Braille image pattern P as input, identifies the positions of the dots within the pattern, maps these positions to a numerical sequence, and returns the numerical sequence representing the Braille image pattern.

3.3.2. Representation of Braille Image as the Location of Dots

Each Braille Cell image is taken and represented by numbers from one through six, as shown in Figure 6. For example, the dots of the first Braille cell image in Figure 11 of Figure 10 are 1236.

Fig. 11 Braille character image

In this step, all the Braille cell images are represented by the location of dots. If there is no dot in the Braille column image, it is considered a space, and a string number sequence of 0020 is added (Unicode of space is 0020). A number sequence of 000A (Unicode of Space) is added when it reaches the end of the row. The number sequence of Fig 10 is shown below.

1236 345 1235 3456 0020 134 345 13456 24 1235 134 4 0020 14 1256 12356 1236 0020 123 134 4 14 26 13456 4 2345 136 0020 0020 0020 0020 0020 0020 0020 0020 0020 000A 1345 345 1235 3456 0020 1345 134 4 1234 24 0020 1345 23456 13 4 13 24 56 4 12456 345 0020 56 26 56 4 12456 26 2345 24 1235 4 0020 0020 0020 0020 000A 1234 1256 1235 3456 0020 1234 1346 12456 4 13 136 23456 134 4 0020 1236 34 2345 4 2345 136 1234 4 0020 1234 136 12456 134 26 346 4 13 136 134 4 0020 000A2345 135 1235 3456 134 4 0020 1345 345 23456 4 23456 13 4 0020 13 56 345 13 4 13 3456 4 23456 16 56 4 0020 2345 135 12356 35 1345 345 56 4 000A

3.4. Creation of Mapping Table

Creating a database to map number strings representing the presence of dot positions to their corresponding language Unicode representations involves several steps. Firstly, define the database schema, typically consisting of a table with columns for the number string and its associated Unicode representation. Next, establish a connection to the database.

Then, each mapping, associating the number string of the Braille character with its corresponding Tamil, Telugu, Malayalam, and Kannada Unicode representation, is inserted into the database table. Finally, close the connection to the database. This algorithmic process ensures the creation of a structured database containing mappings that facilitate easy retrieval of Unicode representations based on input number strings.

3.4.1. Mapping Table Algorithm

Let number_string_unicode_mapping be a dictionary containing mappings between number strings and their Unicode representations in Tamil, Telugu, Malayalam and Kannada.

Create a Mapping look-up Table 1, comprising five fields, holds entries for number strings alongside their corresponding Unicode representations of Consonant, Vowel, and Consonant-Vowel Combined Characters in Tamil, Telugu, Malayalam, and Kannada language.

1. Connect to the database named number_string_unicode_mapping.
2. Create a table named unicode_mapping with columns number_string of Braille
3. Character and unicode_representation for Tamil, Telugu, Malayalam, Kannada
4. For each (number_string, Unicode representations) r in number string Unicode mapping
   - Insert the values into the unicode_mapping table.
5. Close the database connection.

Table 1. Mapping table (braille character image number string associated with its corresponding tamil, telugu, malayalam and kannada unicode)

Braille dot Position	Tamil Unicode	Telugu Unicode	Malayalam Unicode	Kannada Unicode	Braille dot Position	Tamil Unicode	Telugu Unicode	Malayalam Unicode	Kannada Unicode
1	0B85	0C05	0D05	0C85	145	NULL	0C26	0D26	0CA6
345	0B86	0C06	0D06	0C86	2346	NULL	0C27	0D27	0CA7
24	0B87	0C07	0D07	0C87	1345	0BA8	0C28	0D28	0CA8
35	0B88	0C08	0D08	0C88	1234	0BAA	0C2A	0D2A	0CAA
125	0B89	0C09	0D09	0C89	235	NULL	0C2B	0D2B	0CAB
1256	0B8A	0C0A	0D0A	0C8A	12	NULL	0C2C	0D2C	0CAC
26	0B8E	0C0E	0D0E	0C8E	45	NULL	0C2D	0D2D	0CAD
16	0B8F	0C0F	0D0F	0C8F	134	0BAE	0C2E	0D2E	0CAE
34	0B90	0C10	0D10	0C90	13456	0BAF	0C2F	0D2F	0CAF
1346	0B92	0C12	0D12	0C92	1235	0BB0	0C30	0D30	0CB0
135	0B93	0C13	0D13	0C93	12456	0BB1	0C31	0D31	0CB1
246	0B94	0C14	0D14	0C94	123	0BB2	0C32	0D32	0CB2
4	0BCD	0C4D	0D4D	0CCD	456	0BB3	0C33	0D33	0CB3
6	0B83	0C03	0D03	0C83	1236	0BB5	0C35	0D35	0CB5
13	0B95	0C15	0D15	0C95	12356	0BB4	NULL	0D34	NULL
46	NULL	0C16	0D16	0C96	56	0BA9	0C02	0D02	0C82
1245	NULL	0C17	0D17	0C97	146	0BB6	0C36	0D36	0C36
126	NULL	0C18	0D18	0C98	12346	0BB7	0C37	0D37	0C37
346	0B99	0C19	0D19	0C99	234	0BB8	0C38	0D38	0C38
14	0B9A	0C1A	0D1A	0C9A	125	0BB9	0C39	0D39	0C39
16	NULL	0C1B	0D1B	0C9B	345	0BBE	0C3E	0D3E	0CBE
245	0B9C	0C1C	0D1C	0C9C	24	0BBF	0C3F	0D3F	0CBF
356	NULL	0C1D	0D1D	0C9D	35	0BC0	0C40	0D40	0CC0
25	0B9E	0C1E	0D1E	0C9E	136	0BC1	0C41	0D41	0CC1
23456	0B9F	0C1F	0D1F	0C9F	1256	0BC2	0C42	0D42	0CC2
2456	NULL	0C20	0D20	0CA0	26	0BC6	0C46	0D46	0CC6
1246	NULL	0C21	0D21	0CA1	16	0BC7	0C47	0D47	0CC7
123456	NULL	0C22	0D22	0CA2	34	0BC8	0C48	0D48	0CC8
3456	0BA3	0C23	0D23	0CA3	1346	0BCA	0C4A	0D4A	0CCA
2345	0BA4	0C24	0D24	0CA4	135	0BCB	0C4B	0D4B	0CCB
1456	NULL	0C25	0D25	0CA5	246	0BCC	0C4C	0D4C	0CCC

3.5. Conversion of the Number String to its Unicode Representation

Let:

- B be the set of number string of Braille character
- U be the set of Unicode characters for the Southern Indian language.
- M be the mapping function that maps each Braille character to its corresponding Unicode character.

The conversion algorithm can be represented as follows:

3.5.1. Conversion of Number String to Unicode Algorithm

a. Define M as a mapping function that maps each Braille number string b_i to its corresponding Unicode character u_i.:
M: B→U

b. The algorithm iterates through each Braille number string and applies the mapping function M to obtain the corresponding Unicode character.

- Let S_U be the resulting Unicode string obtained after conversion.
- The conversion algorithm can be represented as:
- $S_U=(M(b_1),M(b_2),...,M(b_n))$
- where $M(b_i)$ represents the Unicode character obtained by applying the mapping function to the i_{th} Braille character.

To convert a number string of Braille characters representing a Southern Indian language into its Unicode representation, a mapping database (Table 1) is utilized. This database consists of a collection of mappings that associate each number string of Braille characters with its corresponding Unicode character for the specific Southern Indian language. The algorithm begins by taking the input Braille number string and iterating through each character. The algorithm consults the mapping database for each Braille character encountered to find its corresponding Unicode character. This Unicode character is then appended to a resulting Unicode string. Once all Braille characters have been processed in this manner, the algorithm produces the final Unicode representation of the input Braille string for the Southern Indian language. This method ensures an accurate and efficient conversion process, enabling seamless translation between Braille and Unicode representations. In this procedure, the previous output is associated with the Unicode representation of the chosen language, which is done using Table 1. Each number string is read sequentially and matched with its corresponding Unicode value. For instance, in Figure 10 (Tamil Braille Image), the initial braille character of the number string '1236' is examined in Table 1, and found its Tamil Unicode as '0BB5'. This mapping procedure continues until the end of the file is reached. The Unicode representation of Figure 10 is given below.

0BB5 0BBE 0BB0 0BA3 0020 0BAE 0BBE 0BAF 0BBF 0BB0 0BAE 0BCD 0020 0B9A 0BC2 0BB4 0BB5 0020 0BB2 0BAE 0BCD 0B9A 0BC6 0BAF 0BCD 0BA4 0BC1 0020 000A 0BA8 0BBE 0BB0 0BA3 0020 0BA8 0BAE 0BCD 0BAA 0BBF 0020 0BA8 0B9F 0B95 0BCD 0B95 0BBF 0BA9 0BCD 0BB1 0BBE 0020 0BA9 0BC6 0BA9 0BCD 0BB1 0BC6 0BA4 0BBF 0BB0 0BCD 0020 000A 0BAA 0BC2 0BB0 0BA3 0020 0BAA 0BCA 0BB1 0BCD 0B95 0BC1 0B9F 0BAE 0BCD 0020 0BB5 0BC8 0BA4 0BCD 0BA4 0BC1 0BAA 0BCD 0020 0BAA 0BC1 0BB1 0BAE 0BC6 0B99 0BCD 0B95 0BC1 0BAE 0BCD 0020 000A 0BA4 0BCB 0BB0 0BA3 0BAE 0BCD 0020 0BA8 0BBE 0B9F 0BCD 0B9F 0B95 0BCD 0020 0B95 0BA9 0BBE 0B95 0BCD 0B95 0BA3 0BCD 0B9F 0BC7 0BA9 0BCD 0020 0BA4 0BCB 0BB4 0BC0 0BA8 0BBE 0BA9 0BCD 0020 000A 0020 000A

3.6. Conversion of the Unicode Representation to Editable Text File.

Conversion of Unicode representation to an editable text file involves a systematic process of interpreting the Unicode characters and rendering them as readable text. This process transforms the numerical Unicode values into their corresponding textual counterparts. Subsequently, these decoded characters are structured and organized into a coherent format to generate a text file.

3.6.1. Unicode Character to Editable Text Conversion Algorithm

Input: U be the set of Unicode characters.
Output: T be the set of editable text characters.

The conversion algorithm can be represented as follows:

1. Given a Unicode string S_U of length n, represented as $S_U=(u_1, u_2,...,u_n)$, where u_i is the i_{th} Unicode character.
2. The algorithm iterates through each Unicode character and decodes it back into its corresponding editable text character.+
3. $S_T$be the resulting editable text string obtained after decoding.

$$S_T=\text{decode}(S_U)$$

The decoding process involves the transformation of a Unicode string, represented as $S_U=(u_1,u_2,...,u_n)$, where each u_i is a Unicode character, into an editable text string S_T. Each Unicode character is decoded back into its corresponding editable text character using the rules defined by the Unicode standard.

This step ensures accurate conversion, maintaining fidelity between the original Unicode representation and the resulting editable text. As each Unicode character is decoded, the corresponding editable text character is appended to a new string. This algorithmic approach ensures the creation of an editable text file containing the converted Unicode characters, facilitating easy editing and manipulation of text data. The tamil text of Figure 7 is given below.

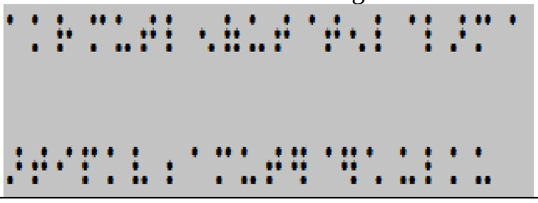
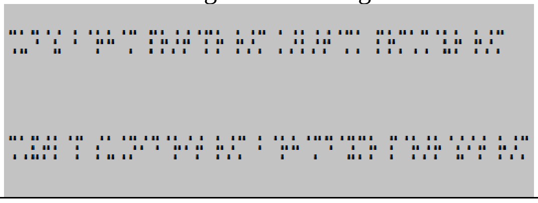
வாரண மாயிரம் சூழவ லம்செய்து
நாரண நம்பி நடக்கின்றா னென்றெதிர்
பூரண பொற்குடம் வைத்துப் புறமெங்கும்
தோரணம் நாட்டக் கனாக்கண்டேன்
தோழீநான்

The proposed system introduced language-specific models for Tamil, Telugu, Kannada, and Malayalam, integrated into a unified framework for multilingual recognition, and this approach set a new benchmark in the field of Braille image conversion systems, significantly improving accessibility and literacy for the visually impaired.

4. Experimental Results

The experimentation of the algorithms was carried out on Tamil, Telugu, Malayalam, and Kannada Braille Image. Some of the experimental results are shown below in Tables 2 and 3.

Table 2. Conversion of braille image to its corresponding language (Tamil & Telugu)

Tamil Braille Image to Editable Text	**Telugu Braille Image to Editable Text**
Tamil Braille Image 	Telugu Braille Image 

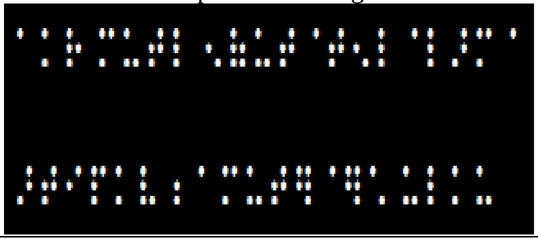
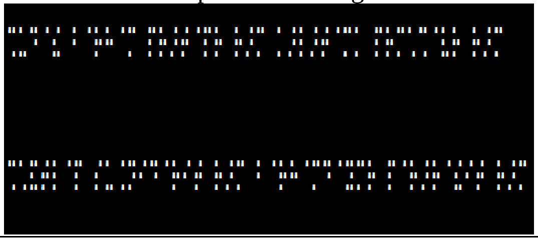
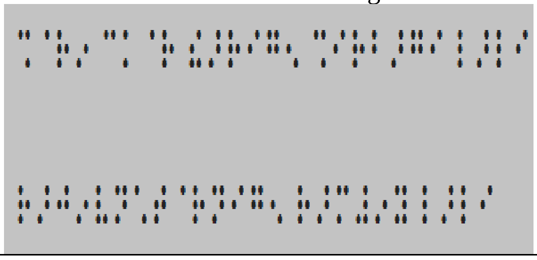
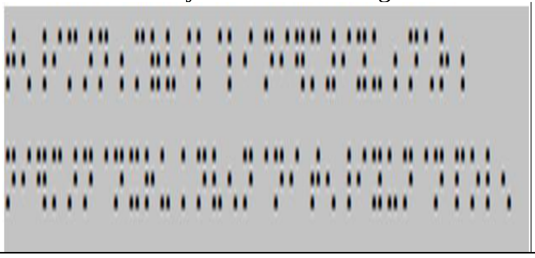
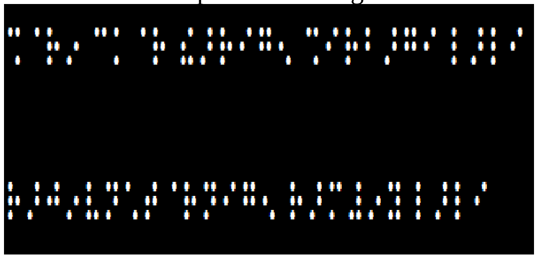
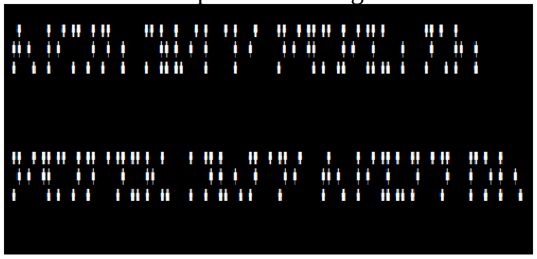
Preprocessed Image 	Preprocessed Image 
Number String Conversion 1 13 1235 0020 134 136 2345 123 0020 26 12356 136 2345 4 2345 26 123 4 123 345 134 4 0020 345 2345 24 000A 1234 13 1236 56 4 0020 134 136 2345 12456 4 12456 16 0020 136 123 13 136 0020 0020 000A	Number String Conversion 146 136 145 4 2346 0020 12 4 1235 125 4 134 0020 1234 1235 345 2345 4 1234 1235 0020 1235 345 134 0020 13 345 123 345 2345 4 134 13 0020 1234 1235 134 16 146 4 1236 1235 0020 1235 345 134 000A 146 16 12346 2345 123 4 1234 0020 234 136 46 1345 24 145 4 1235 24 2345 0020 1235 345 134 0020 12 4 1235 125 4 134 145 4 13456 134 1235 0020 1234 4 1235 345 1235 4
Tamil Unicode Conversion 0B85 0B95 0BB0 0020 0BAE 0BC1 0BA4 0BB2 0020 0B8E 0BB4 0BC1 0BA4 0BCD 0BA4 0BC6 0BB2 0BCD 0BB2 0BBE 0BAE 0BCD 0020 0B86 0BA4 0BBF 000A 0BAA 0B95 0BB5 0BA9 0BCD 0020 0BAE 0BC1 0BA4 0BB1 0BCD 0BB1 0BC7 0020 0B89 0BB2 0B95 0BC1 000A	Telugu Unicode Conversion 0C36 0C41 0C26 0C4D 0C27 0020 0C2C 0C4D 0C30 0C39 0C4D 0C2E 0020 0C2A 0C30 0C3E 0C24 0C4D 0C2A 0C30 0020 0C30 0C3E 0C2E 0020 0C15 0C3E 0C32 0C3E 0C24 0C4D 0C2E 0C15 0020 0C2A 0C30 0C2E 0C47 0C36 0C4D 0C35 0C30 0020 0C30 0C3E 0C2E 000A 0C36 0C47 0C37 0C24 0C32 0C4D 0C2A 0020 0C38 0C41 0C16 0C28 0C3F 0C26 0C4D 0C30 0C3F 0C24 0020 0C30 0C3E 0C2E 0020 0C2C 0C4D 0C30 0C39 0C4D 0C2E 0C26 0C4D 0C2F 0C2E 0C30 0020 0C2A 0C4D 0C30 0C3E 0C30 0C4D 0C27 0C3F 0C24 0020 0C30 0C3E 0C2E 000A
Editable Tamil Text அகர முதல எழுத்தெல்லாம் ஆதி பகவன் முதற்றே உலகு	Editable Telugu Text శుద్ధ బ్రహ్మ పరాత్పర రామ కాలాత్మక పరమేశ్వర రామ శేషతల్ప సుఖనిద్రిత రామ బ్రహ్మద్యమర ప్రార్ధిత రామ

Table 3. Conversion of braille image to its corresponding language (Kannada & Malayalam)

Kannada Braille Image to Editable Text	Malayalam Braille Image to Editable Text
Kannada Braille Image 	Malayalam Braille Image 
Preprocessed Image 	Preprocessed Image 

Number String Conversion 146 4 1235 35 0020 14 13 4 1235 0020 2346 345 1235 24 1245 26 0020 146 24 1235 12 345 1245 24 0020 123 345 123 24 000A 1235 345 245 35 1236 1345 16 2345 4 1235 1345 24 1245 26 0020 1235 345 134 3456 35 13456 0020 123 345 123 24 0020 0020 000A	Number String Conversion 2345 26 456 24 134 345 1345 56 0020 134 12356 1236 24 123 4 123 24 0020 1345 24 12456 134 3456 24 13456 136 56 0020 1345 16 1235 56 000A 1345 24 12456 134 345 1345 4 1345 1346 1235 136 0020 13 1345 1236 26 1345 4 1345 24 0020 2345 26 456 24 13456 136 1345 4 1345 0020 1234 135 123 26 000A
Kannada Unicode Conversion 0CB6 0CCD 0CB0 0CC0 0020 0C9A 0C95 0CCD 0CB0 0020 0CA7 0CBE 0CB0 0CBF 0C97 0CC6 0020 0CB6 0CBF 0CB0 0CAC 0CBE 0C97 0CBF 0020 0CB2 0CBE 0CB2 0CBF 000A 0CB0 0CBE 0C9C 0CC0 0CB5 0CA8 0CC7 0CA4 0CCD 0CB0 0CA8 0CBF 0C97 0CC6 0020 0CB0 0CBE 0CAE 0CA3 0CC0 0CAF 0020 0CB2 0CBE 0CB2 0CBF 000A	Malayalam Unicode Conversion 0D24 0D46 0D33 0D3F 0D2E 0D3E 0D28 0D02 0020 0D2E 0D34 0D35 0D3F 0D32 0D4D 0D32 0D3F 0D7B 0020 0D28 0D3F 0D31 0D2E 0D23 0D3F 0D2F 0D41 0D02 0020 0D28 0D47 0D30 0D02 000A 0D28 0D3F 0D31 0D2E 0D3E 0D7C 0D28 0D4D 0D28 0D4A 0D30 0D41 0020 0D15 0D28 0D35 0D46 0D28 0D4D 0D28 0D3F 0D7D 0020 0D24 0D46 0D33 0D3F 0D2F 0D41 0D28 0D4D 0D28 0020 0D2A 0D4B 0D32 0D46 000A 0020 0CB0 0CBE 0CAE 0CA3 0CC0 0CAF 0020 0CB2 0CBE 0CB2 0CBF 000A
Editable Kannada Text ಶ್ರೀ ಚಕ್ರ ಧಾರಿಗೆ ಶಿರಬಾಗಿ ಲಾಲಿ ರಾಜೀವನೇತ್ರನಿಗೆ ರಾಮಣೀಯ ಲಾಲಿ	Editable Malayalam Text തെളിമാനം മഴവില്ലിൻ നിറമണിയും നേരം നിറമാർന്നൊരു കനവെന്നിൽ തെളിയുന്ന പോലെ

Experimental results show that the proposed method has a good performance in converting any southern Indian Braille into its language script.

5. Conclusion

The utilization of the Braille system for reading and writing has been long-standing among individuals with visual impairments. This paper presents a novel approach aimed at converting scanned Braille documents in four southern Indian languages into editable text files. The process initiates with preprocessing the Braille documents to enhance the dots and reduce noise interference. Following this, the Braille cells undergo segmentation, and the dots within each cell are extracted and converted into a location sequence. These sequences are then mapped to the corresponding alphabets of the respective language. The resulting converted text can further be synthesized into speech using any speech synthesizer. Experimental results demonstrate the effectiveness of the proposed method in accurately converting Braille script to text. The research successfully translates Braille scripts from Tamil, Telugu, Kannada, and Malayalam into their corresponding languages, giving potential extensions the ability to recognize and convert other Indian Braille script languages in the future. This development marks a significant stride towards improving accessibility and inclusivity for visually impaired individuals in southern India. By accurately interpreting Braille characters in native languages such as Tamil, Telugu, Kannada, and Malayalam, this system addresses the crucial need for effective tools to access written information.

References

[1] Barry Hampshire, *Working with Braille: A Study of Braille as a Medium of Communication*, The UNESCO Press, 1981. [Google Scholar] [Publisher Link]

[2] Jan Mennens, "Optical Recognition of Braille Writing," *Proceedings of 2nd International Conference on Document Analysis and Recognition*, Tsukuba, Japan, pp. 428-431, 1993. [CrossRef] [Google Scholar] [Publisher Link]

[3] Jan Mennens, "Optical Recognition of Braille Writing Using Standard Equipment," *IEEE Transactions on Rehabilitation Engineering*, vol. 2, no. 4, pp. 207-212, 1994. [CrossRef] [Google Scholar] [Publisher Link]

[4] Braille, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Braille

[5] Specification 800:2014 Braille Books and Pamphlets, National Library Service for the Blind and Physically Handicapped, Library of Congress, 2014. [Online]. Available: https://www.loc.gov/nls/wp-content/uploads/2019/09/Spec800.11October2014.final_.pdf

[6] Bharati Braille, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Bharati_Braille

[7] Andrean Ignasius et al., “Image Pre-Processing Effect on OCR's Performance for Image Conversion to Braille Unicode,” *Procedia Computer Science*, vol. 227, pp. 922-931, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[8] Sana Shokat et al., “Characterization of English Braille Patterns Using Automated Tools and RICA Based Feature Extraction Methods,” *Sensors*, vol. 22, no. 5, pp. 1-23, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[9] Falgoon Sen Apu et al., “Text and Voice to Braille Translator for Blind People,” *2021 International Conference on Automation, Control and Mechatronics for Industry 4.0*, Rajshahi, Bangladesh, pp. 1-6, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[10] S. Arockia Kinsely Felix et al., “Enhancing Braille Code Conversion to Text in Multiple Languages,” *i-Manager's Journal on Digital Signal Processing*, vol. 8, no. 2, pp. 31-36, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[11] Purvang Patel, V.V. Hanchate, and R.S. Kamathe, “Text To Braille Conversion For Real-Time Teaching (For Grade III Braille),” *Natural Volatiles & Essential Oils Journal*, vol. 8, no. 6, pp. 3939-3945, 2024. [Publisher Link]

[12] Ghazanfar Latif et al., “Learning at Your Fingertips: An Innovative IoT-Based AI-Powered Braille Learning System,” *Applied System Innovation*, vol. 6, no. 5, pp. 1-18, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[13] Sana Shokat, “Analysis and Evaluation of Braille to Text Conversion Methods,” *Mobile Information Systems*, vol. 2020, pp. 1-14, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[14] Changjian Li, and Weiqi Yan, “Braille Recognition Using Deep Learning,” *Proceedings of the 4th International Conference on Control and Computer Vision*, Macau China, pp. 30-35, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[15] Robert Englebretson, M. Cay Holbrook, and Simon Fischer-Baum, “A Position Paper on Researching Braille in the Cognitive Sciences: Decentering the Sighted Norm,” *Applied Psycholinguistics*, vol. 44, no. 3, pp. 400-415, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[16] J.M. Jennis Oliviya, J. Vijayakumar, and A. Hemalatha, “Conversion of Tamil Braille into Text Information Using the DCNN Technique,” *International Journal of All Research Education and Scientific Methods*, vol. 11, no. 4, pp. 575-579, 2023. [Publisher Link]

[17] Ian Herrera, Juan J. Carreras, and Rutilio Nava, “Voice-to-Braille Translation System for Promoting Braille Learning,” *International Journal of Engineering and Technology*, vol. 16, no. 1, pp. 52-56, 2024. [Publisher Link]

[18] Urooj Beg, K. Parvathi, and Vinod Jha, “Text Translation of Scanned Hindi Document to Braille via Image Processing,” *Indian Journal of Science and Technology*, vol. 10, no. 33, pp. 1-8, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[19] Ganga Gudi, Mallamma V. Reddy, and M. Hanumanthappa, “Efficient Approach for Braille Conversion of Multilingual Text: Strategic Mapping Solutions,” *International Journal of Engineering Technology and Management Sciences*, vol. 7, no. 4, pp. 653-659, 2023. [CrossRef] [Publisher Link]

[20] Meneses-Claudio, W. Alvarado-Diaz, and A. Roman-Gonzalez, “Classification System for the Interpretation of the Braille Alphabet through Image Processing,” *Advances in Science, Technology and Engineering Systems Journal*, vol. 5, no. 1, pp. 403-407, 2020. [CrossRef] [Google Scholar] [Publisher Link]