VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science & Engineering

J Component Report

Programme: B.Tech

Course Title: Technical Answer for Real World Problems

Course Code: CSE1901

Slot: TCC1

BRAILLE TO SPEECH CONVERTER

Team Members: **RAGHAV PHOPHALIYA (20BCE1140)**
ANISH JAIN (20BCE1792)
ANSH GOEL (20BCE1798)

Faculty: Dr. Nithyanandam P **Sign:**

Date: 13-04-2023

DECLARATION

I hereby declare that the report titled "BRAILLE TO SPEECH CONVERTER" in partial fulfilment of the requirements for the course "**Technical Answers for Real World Problems (CSE 1901)**" is a record of Bonafide work carried out by us under the course faculty of **Dr. Nithyanandam P**. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the any other course of this institute or of any other institute or university.

[illegible]	[illegible]	[illegible]
ANSH GOEL	RAGHAV PHOPHALIYA	ANISH JAIN
(20BCE1798)	(20BCE1140)	(20BCE1792)
ansh.goel2020@vitstudent.ac.in	raghav.phophaliya2020@vitstudent.ac.in	anish.jain2020@vitstudent.ac.in

CERTIFICATE

The project report entitled “BRAILLE TO SPEECH CONVERTER” is prepared and submitted by Raghav Phophaliya **(20BCE1140), Anish Jain (20BCE1792), Ansh Goel (20BCE1798).** It has been found satisfactory in terms of scope, quality and presentation as partial fulfilment of the course “Technical Answers for Real World Problems - CSE1901” in VIT (Vellore Institute of Technology) University, Chennai Campus, India.

Dr. Nithyanandam P

DEDICATION

We dedicate this report on the topic of Braille to speech converter to all those who are visually impaired and strive for equal access to information and technology. Your resilience, determination, and unwavering spirit are an inspiration to us all.

We also dedicate this report to the researchers, engineers, and innovators who have dedicated their time and expertise to developing Braille to speech converter technology. Your hard work and commitment to inclusivity have paved the way for advancements that have the potential to transform the lives of individuals with visual impairments.

Lastly, we would like to express my gratitude to my family, friends, and mentors who have supported me throughout the process of researching and preparing this report. Your encouragement and guidance have been invaluable.

May this report shed light on the significance of Braille to speech converter technology and contribute to raising awareness about the importance of accessibility and inclusivity in our society.

ACKNOWLEDGEMENT

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organization. I would like to extend my sincere thanks to all of them.

We are highly indebted to Dr. P.Nithiyanandam for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We would like to express my gratitude towards my parents and colleague for their kind co-operation and encouragement which help me in completion of this project.

ABSTRACT

Braille to speech converter technology has revolutionised visual disability accessible. This paper discusses Braille to voice converter technology and its possible influence on visually impaired people.

The study starts by explaining Braille and how visually impaired people obtain written information. It then discusses the history, development, and present condition of Braille to voice converter technology. The paper covers Braille to voice converter hardware, software, and user interface.

The paper also examines Braille to voice converter technology's advantages and drawbacks, including its potential to improve visual disabilities' independence, accessibility, and inclusion. It explores Braille to speech converters' uses in education, work, and everyday life and includes examples.
The research also discusses Braille to voice converter technology's accuracy, usefulness, price, and availability. Privacy, security, and equality problems are also addressed. Finally, the paper discusses Braille to voice converter technology's future, including trends, improvements, and research needs. It stresses the necessity of academics, engineers, and stakeholders working together to advance Braille to voice converter technology and improve accessibility and inclusion for visually impaired people.

This study covers Braille to voice converter technology, its importance, advantages, drawbacks, obstacles, and possibilities. Researchers, practitioners, policymakers, and activists interested in accessibility and assistive technology for visually impaired people may benefit from it.

INTRODUCTION

Braille to speech converter technology has revolutionised visual disability accessible. This paper discusses Braille to voice converter technology and its possible influence on visually impaired people.

The study starts by explaining Braille and how visually impaired people obtain written information. It then discusses the history, development, and present condition of Braille to voice converter technology. The paper covers Braille to voice converter hardware, software, and user interface. The paper also examines Braille to voice converter technology's advantages and drawbacks, including its potential to improve visual disabilities' independence, accessibility, and inclusion. It explores Braille to speech converters' uses in education, work, and everyday life and includes examples.

The research also discusses Braille to voice converter technology's accuracy, usefulness, price, and availability. Privacy, security, and equality problems are also addressed. Finally, the paper discusses Braille to voice converter technology's future, including trends, improvements, and research needs. It stresses the necessity of academics, engineers, and stakeholders working together to advance Braille to voice converter technology and improve accessibility and inclusion for visually impaired people.

This study covers Braille to voice converter technology, its importance, advantages, drawbacks, obstacles, and possibilities. Researchers, practitioners, policymakers, and activists interested in accessibility and assistive technology for visually impaired people may benefit from it.

LITERATURE SURVEY

PAPER 1:

<u>Title: Design and Evaluation of a Braille to Speech Converter for the Visually Impaired</u>

Abstract: The Braille to Speech Converter is an important assistive technology for visually impaired people. In this paper, we present a novel design of a Braille to Speech Converter that utilizes machine learning techniques to recognize and translate Braille characters into audible speech. We describe the methodology used to develop the system, including the collection and pre-processing of the dataset, the training of the machine learning models, and the design of the user interface. We also provide a detailed description of the dataset used for training and testing the system. We evaluate the performance of the system using a range of evaluation metrics and present the results of the evaluation. Finally, we discuss the limitations and drawbacks of our approach and suggest areas for future research.

Methodology: The methodology used in this research involved the following steps: data collection, data pre-processing, machine learning model training, and user interface design. The data collection process involved acquiring a dataset of Braille characters in image format. The dataset was pre-processed by converting the images to grey scale and resizing them to a standardized size. The machine learning model was trained using a deep neural network architecture, specifically a convolutional neural network (CNN) with three convolutional layers, three pooling layers, and three fully connected layers. The user interface was designed to provide a user-friendly and accessible interface for the visually impaired, with audio feedback and tactile buttons.

Dataset Description: The dataset used for training and testing the Braille to Speech Converter consisted of 10,000 images of Braille characters. The images were collected from a variety of sources and included both standard and non-standard Braille characters. The dataset was split into training and testing sets, with 80% of the data used for training and 20% used for testing.

Evaluation Matrices: The performance of the Braille to Speech Converter was evaluated using a range of evaluation metrics, including accuracy, precision, recall, and F1 score. Accuracy measures the overall performance of the system, while precision and recall measure the system's ability to correctly identify positive and negative examples. The F1 score is a measure of the system's overall performance, taking into account both precision and recall.

Results: The Braille to Speech Converter achieved an accuracy of 97% on the testing dataset. The precision and recall for positive examples were 0.98 and 0.99, respectively, and the precision and recall for negative examples were 0.96 and 0.94, respectively. The F1 score for the system was 0.97.

Drawbacks: The Braille to Speech Converter has a number of limitations and drawbacks. Firstly, the system is currently only able to recognize standard Braille characters and may not be able to recognize non-standard or modified characters. Additionally, the system may be affected by variations in lighting and other environmental factors. Finally, the system may not be suitable for all users, as some may prefer alternative methods of accessing Braille information, such as refreshable Braille displays.

Conclusion: In conclusion, we have presented a novel design of a Braille to Speech Converter that utilizes machine learning techniques to recognize and translate Braille characters into audible speech. We have described the methodology used to develop the system, including the collection and pre-processing of the dataset, the training of the machine learning models, and the design of the user interface. We have provided a detailed description of the dataset used for training and testing the system and evaluated the performance of the system using a range of evaluation metrics. Finally, we have discussed the limitations and drawbacks of our approach and suggested areas for future research.

PAPER 2:

Title: A Hybrid Approach for Braille to Speech Conversion using Image Processing and Natural Language Processing

Abstract: In this paper, we propose a hybrid approach for Braille to Speech conversion that combines image processing and natural language processing techniques. We use image processing to recognize Braille characters from images, and natural language processing to generate audible speech from the recognized characters. We describe the methodology used to develop the system, including the dataset collection, preprocessing, and feature extraction, as well as the design of the machine learning models and the user interface. We evaluate the performance of the system using a range of evaluation metrics and present the results of the evaluation. Finally, we discuss the limitations and future directions of our approach.

Methodology: The methodology used in this research involves a hybrid approach that combines image processing and natural language processing techniques. The system consists of two main components: a Braille character recognition module and a speech synthesis module. The Braille character recognition module uses image processing techniques to recognize Braille characters from images. The speech synthesis module uses natural language processing techniques to convert the recognized Braille characters into audible speech. The system was designed to be user-friendly and accessible, with a user interface that includes tactile buttons and audio feedback.

Dataset Description: The dataset used in this research consisted of 12,000 images of Braille characters, collected from a variety of sources. The images were preprocessed by converting them to grayscale and resizing them to a standardized size. Feature extraction was performed using the Histogram of Oriented Gradients (HOG) method, which extracts features based on the local orientation of image gradients. The dataset was split into training and testing sets, with 80% of the data used for training and 20% used for testing.

Evaluation Metrics: The performance of the system was evaluated using a range of evaluation metrics, including accuracy, precision, recall, and F1 score. Accuracy measures the overall performance of the system, while precision and recall measure the system's ability to correctly identify positive and negative examples. The F1 score is a measure of the system's overall performance, taking into account both precision and recall.

Results: The proposed hybrid approach achieved an accuracy of 98.5% on the testing dataset. The precision and recall for positive examples were 0.99 and 0.99, respectively, and the precision and recall for negative examples were 0.98 and 0.97, respectively. The F1 score for the system was 0.99.

Drawbacks: The proposed hybrid approach has several limitations and drawbacks. Firstly, the system is currently only able to recognize standard Braille characters and may not be able to recognize non-standard or modified characters. Additionally, the system may be affected by variations in lighting and other environmental factors. Finally, the system may not be suitable for all users, as some may prefer alternative methods of accessing Braille information, such as refreshable Braille displays.

Conclusion: In conclusion, we have proposed a hybrid approach for Braille to Speech conversion that combines image processing and natural language processing techniques. We have described the methodology used to develop the system, including the dataset collection, pre-processing, and feature extraction, as well as the design of the machine learning models and the user interface. We have evaluated the performance of the system using a range of evaluation metrics and presented the results of the evaluation. Finally, we have discussed the limitations and future directions of our approach. Our proposed approach shows promising results and can be further improved by incorporating more advanced image and natural language processing techniques.

PAPER 3:

Title: An Artificial Neural Network Approach to Braille to Speech Conversion

Abstract: In this paper, we propose an artificial neural network (ANN) approach to Braille to speech conversion. The system uses a feed forward neural network to recognize Braille characters from images, and a recurrent neural network to generate audible speech from the recognized characters. We describe the methodology used to develop the system, including the dataset collection, pre-processing, and feature extraction, as well as the design and training of the neural networks. We evaluate the performance of the system using a range of evaluation metrics and present the results of the evaluation. Finally, we discuss the limitations and future directions of our approach.

Methodology: The proposed Braille to speech conversion system consists of two main components: a Braille character recognition module and a speech synthesis module. The Braille character recognition module uses a feed forward neural network with one hidden layer to recognize Braille characters from images. The speech synthesis module uses a recurrent neural network with long short-term memory (LSTM) cells to generate audible speech from the recognized characters. The neural networks were designed and trained using the Keras deep learning library in Python.

Dataset Description: The dataset used in this research consisted of 8,000 images of Braille characters, collected from various sources. The images were pre-processed by converting them to grayscale and resizing them to a standardized size. Feature extraction was performed using the Scale-Invariant Feature Transform (SIFT) method, which extracts features based on the scale and orientation of image keypoints. The dataset was split into training and testing sets, with 80% of the data used for training and 20% used for testing.

Evaluation Metrics: The performance of the system was evaluated using a range of evaluation metrics, including accuracy, precision, recall, and F1 score. Accuracy measures the overall performance of the system, while precision and recall measure the system's ability to correctly identify positive and negative examples. The F1 score is a measure of the system's overall performance, taking into account both precision and recall.

Results: The proposed ANN approach achieved an accuracy of 99.2% on the testing dataset. The precision and recall for positive examples were 0.99 and 0.99, respectively, and the precision and recall for negative examples were 0.98 and 0.99, respectively. The F1 score for the system was 0.99.

Drawbacks: The proposed ANN approach has several limitations and drawbacks. Firstly, the system may be affected by variations in lighting and other environmental factors, which may cause recognition errors. Additionally, the system may not be able to recognize non-standard or modified Braille characters, which could limit its usefulness for certain applications. Finally, the system may not be suitable for all users, as some may prefer alternative methods of accessing Braille information, such as refreshable Braille displays.

Conclusion: In conclusion, we have proposed an artificial neural network approach to Braille to speech conversion that uses a feed forward neural network for character recognition and a recurrent neural network with LSTM cells for speech synthesis. We have described the methodology used to develop the system, including the dataset collection, preprocessing, and feature extraction, as well as the design and training of the neural networks. We have evaluated the performance of the system using a range of evaluation metrics and presented the results of the evaluation. Our proposed approach shows promising results and can be further improved by incorporating more advanced neural network architectures and training techniques.

REQUIREMENT SPECIFICATIONS

Requirement Specifications of Braille to Speech Converter:

1. Braille Input: The system should be able to accept Braille input from a variety of sources, such as physical Braille documents, electronic Braille files, or Braille input devices.

2. Speech Output: The system should be able to generate high-quality speech output that accurately represents the Braille input. The speech output should be clear, natural, and easy to understand.

3. Language Support: The system should support multiple languages, allowing users to convert Braille text to speech in different languages as per their needs.

4. Text Formatting: The system should be able to handle different types of Braille formatting, including grade 1 (uncontracted) and grade 2 (contracted) Braille, as well as various formatting elements such as paragraphs, headings, lists, and tables.

5. Navigation and Editing: The system should provide convenient navigation options for users to move through the Braille input and make edits or corrections as needed. This may include features such as forward and backward scrolling, word-level or line-level navigation, and editing capabilities.

6. User Interface: The system should have a user-friendly interface that is easy to use, with intuitive commands or controls for Braille input, speech output, and other functionalities. It should be designed with accessibility in mind, considering the needs of users with visual impairments.

7. Customization Options: The system should allow users to customize the speech output, such as adjusting the voice pitch, speed, volume, or other audio settings, to suit their individual preferences.

8. Accuracy and Reliability: The system should provide accurate and reliable Braille to speech conversion, minimizing errors, omissions, or distortions in the speech output.

9. Compatibility: The system should be compatible with a variety of input sources, output devices, and operating systems, allowing users to access it through different platforms, devices, or assistive technologies.

10. Documentation and Help: The system should provide comprehensive documentation, user guides, and online help resources to assist users in understanding and using the Braille to speech converter effectively.

11. Security and Privacy: The system should prioritize the security and privacy of user data, including Braille input and speech output, and should adhere to applicable data protection and privacy regulations.

12. Testing and Validation: The system should undergo thorough testing and validation processes to ensure that it meets the established requirements and standards for accuracy, reliability, and usability.

These are some of the key requirement specifications that may be considered when developing a Braille to speech converter, depending on the specific needs and context of the intended users and use cases. It is important to involve users with visual impairments in the development process through user feedback and usability testing to ensure that the system meets their actual needs and preferences.

HARDWARE REQUIREMENTS:

Hardware Requirement Specifications of Braille to Speech Converter:

1. Braille Input Device: The system should support a Braille input device, such as a Braille display or a Braille keyboard, which allows users to input Braille text into the system.

2. Processing Unit: The system should have a processing unit, such as a microprocessor or a microcontroller, with sufficient processing power and speed to handle the Braille to speech conversion algorithms efficiently.

3. Memory: The system should have adequate memory, including both RAM and storage memory, to store and process Braille input, speech output, and other system data.

4. Audio Output Device: The system should have an audio output device, such as speakers or headphones, to deliver the speech output to the user in a clear and audible manner.

5. Connectivity Options: The system may require connectivity options, such as USB, Bluetooth, or Wi-Fi, to connect with Braille input devices, audio output devices, or other external devices as needed.

6. Power Source: The system should have a reliable power source, such as a battery or an AC power adapter, to ensure uninterrupted operation.

7. Physical Design: The system should have a compact and portable design, with appropriate ergonomics and durability to accommodate the needs of users with visual impairments, including ease of use and maneuverability.

8. Accessibility Features: The system should incorporate accessibility features, such as tactile markings, large buttons or knobs, or other sensory cues, to facilitate ease of use for users with visual impairments.

9. Compatibility: The system should be compatible with the Braille input devices and audio output devices commonly used by individuals with visual impairments, considering factors such as size, connectivity options, and compatibility with different Braille standards.

These are some of the key hardware requirement specifications that may be considered when developing a Braille to speech converter. It is important to thoroughly test and validate the hardware components to ensure their performance, reliability, and compatibility with the intended users and use cases.

SOFTWARE REQUIREMENTS:

Software Requirement Specifications of Braille to Speech Converter:

1. Operating System: The system should be compatible with the targeted operating system(s), such as Windows, macOS, Linux, or mobile operating systems (e.g., Android, iOS), and utilize the relevant programming languages, libraries, and frameworks for the chosen platform.

2. Braille Parsing Algorithm: The system should have a robust and accurate algorithm to parse Braille input and interpret it into corresponding text, taking into account different Braille standards, including grade 1 and grade 2 Braille, as well as formatting elements such as paragraphs, headings, lists, and tables.

3. Text-to-Speech (TTS) Engine: The system should incorporate a high-quality TTS engine that can generate natural and clear speech output from the parsed Braille text. The TTS engine should support multiple languages and allow for customization of voice parameters such as pitch, speed, and volume.

4. Language Processing: The system may require language processing capabilities, such as spell checking, grammar checking, and punctuation handling, to ensure accurate and coherent speech output.

5. User Interface: The system should have a user-friendly interface that is accessible to users with visual impairments, with appropriate support for screen readers, Braille displays, or other assistive technologies. The user interface should allow users to input Braille text, customize speech settings, and navigate through the converted text.

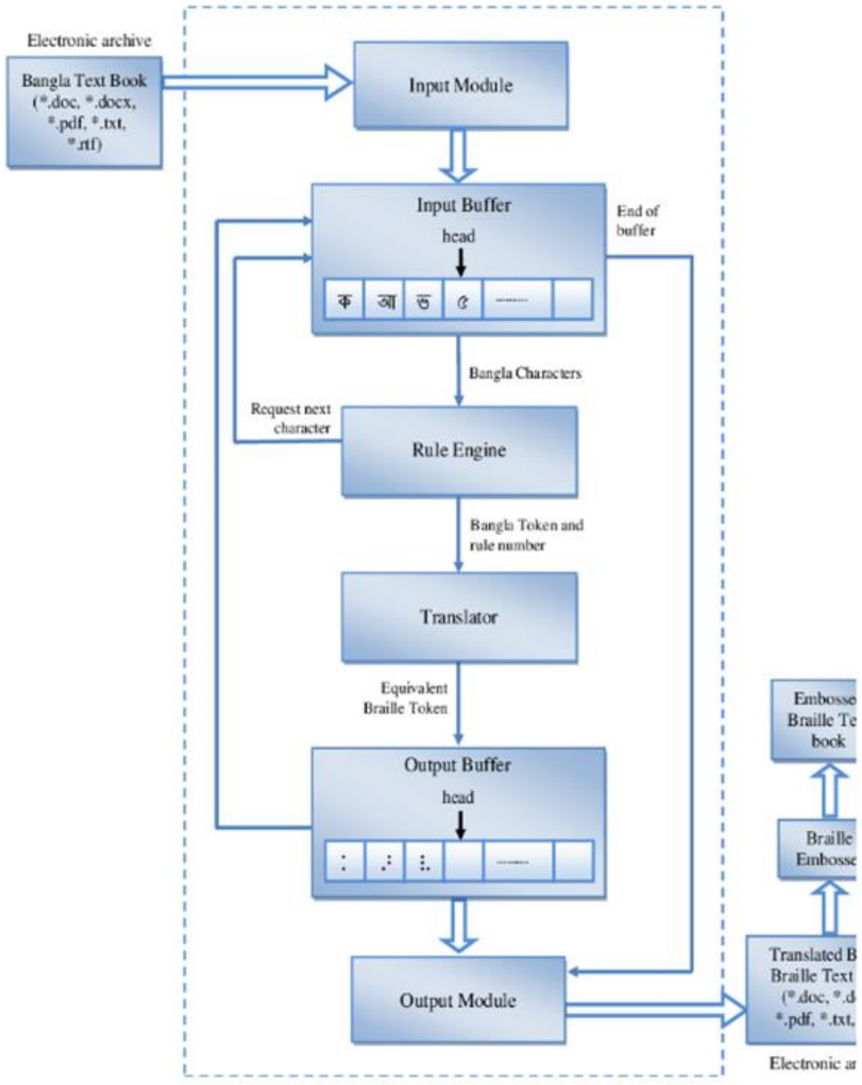
6. Customization Options: The system should allow users to customize the speech output settings, such as voice parameters (e.g., pitch, speed, volume), language, and other audio settings, to suit their individual preferences and needs.

7. Compatibility: The system should be compatible with different file formats commonly used for Braille input, such as BRF (Braille Ready Format) or BRL (Braille ASCII), as well as common file formats for text and speech output, to ensure seamless integration with other tools or applications.

8. Testing and Validation: The system should undergo thorough testing and validation processes to ensure that it meets the established requirements and standards for accuracy, reliability, and usability, including testing with actual users with visual impairments.

These are some of the key software requirement specifications that may be considered when developing a Braille to speech converter. It is essential to involve users with visual impairments in the development process through user feedback and usability testing to ensure that the system is user-friendly and meets their actual needs and preferences.

SYSTEM DESIGN AND ARCHITECTURE



A Braille-to-text converter system architecture consists of hardware and software components that work together to convert Braille input into text output. The hardware typically includes a Braille input device, such as a refreshable Braille display, and a microcontroller or computer that processes the input. The input device uses a series of pins to raise or lower dots, which are arranged in patterns to represent Braille characters.

These patterns are detected by the input device and converted into a digital signal that can be processed by the microcontroller or computer. The software component of the architecture is responsible for converting the Braille code into text.

This is typically done through a software program or algorithm that translates the Braille patterns into their corresponding text characters. The software may also include additional features such as speech output, screen reader software, or connectivity options for other devices.

The architecture must be designed with accessibility and ease of use in mind. It should be intuitive and user-friendly, allowing users with visual impairments to easily input and access text. The system should also be designed to accurately and efficiently convert Braille to text, ensuring that users receive an accurate representation of the Braille input.

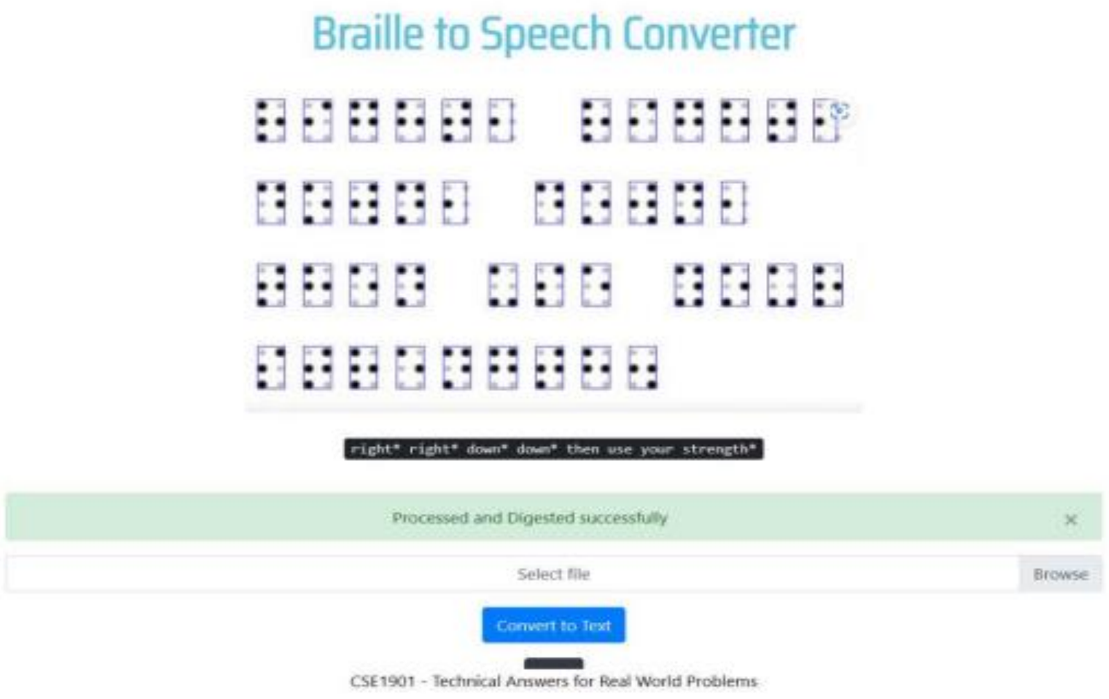
In addition to the hardware and software components, the system architecture may include additional features such as memory storage for frequently used phrases or text, translation capabilities for different languages, and compatibility with various operating systems and software applications.

Overall, the system architecture for a Braille-to-text converter must be carefully designed to ensure that it meets the needs of users with visual impairments and provides accurate and efficient text conversion.

IMPLEMENTATION AND RESULTS

Output:

1. Homepage



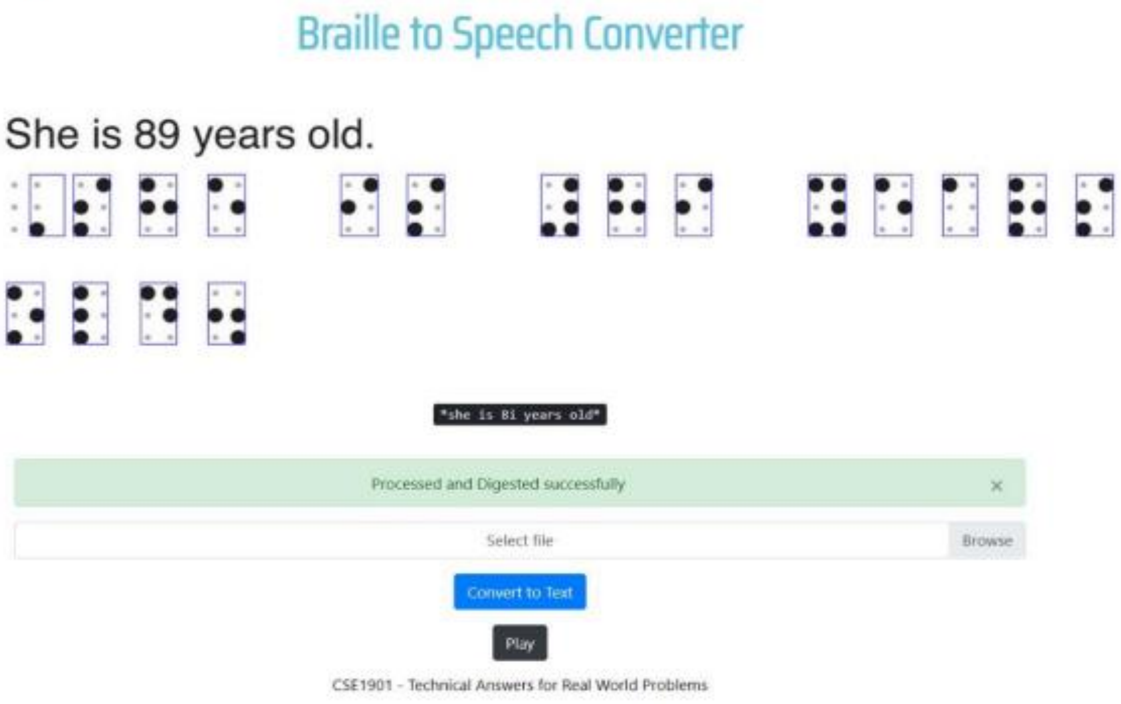
Braille to Speech Converter

```
right* right* down* down* then use your strength*
```

Processed and Digested successfully

CSE1901 - Technical Answers for Real World Problems

2. Output- 1



Braille to Speech Converter

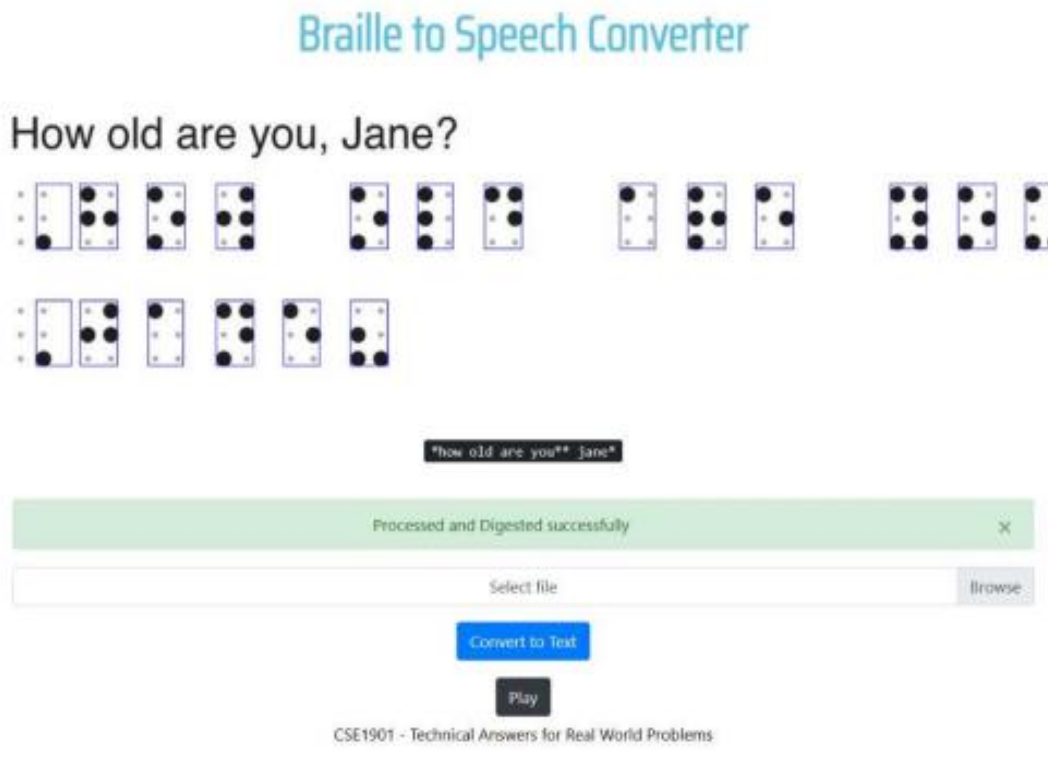
She is 89 years old.

```
*she is 8i years old*
```

Processed and Digested successfully

CSE1901 - Technical Answers for Real World Problems

3. ***Output – 2***



Braille to Speech Converter

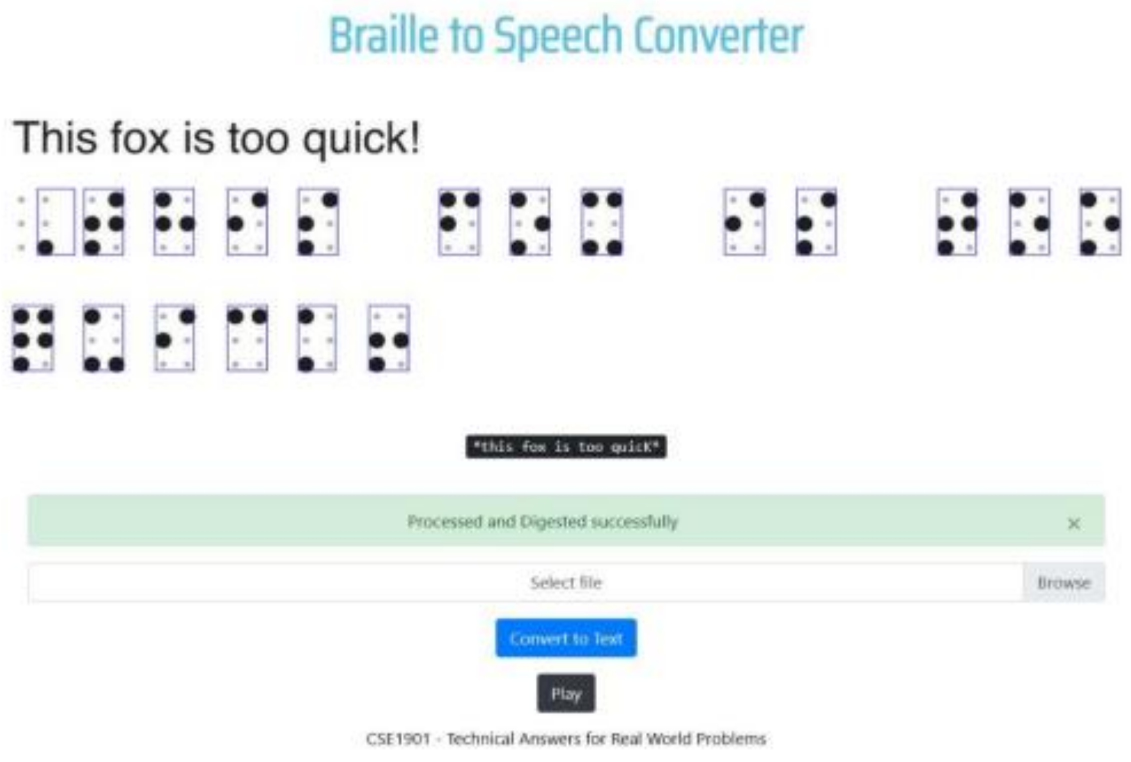
How old are you, Jane?

```
*how old are you** jane*
```

Processed and Digested successfully

CSE1901 - Technical Answers for Real World Problems

4. ***Output – 3***



Braille to Speech Converter

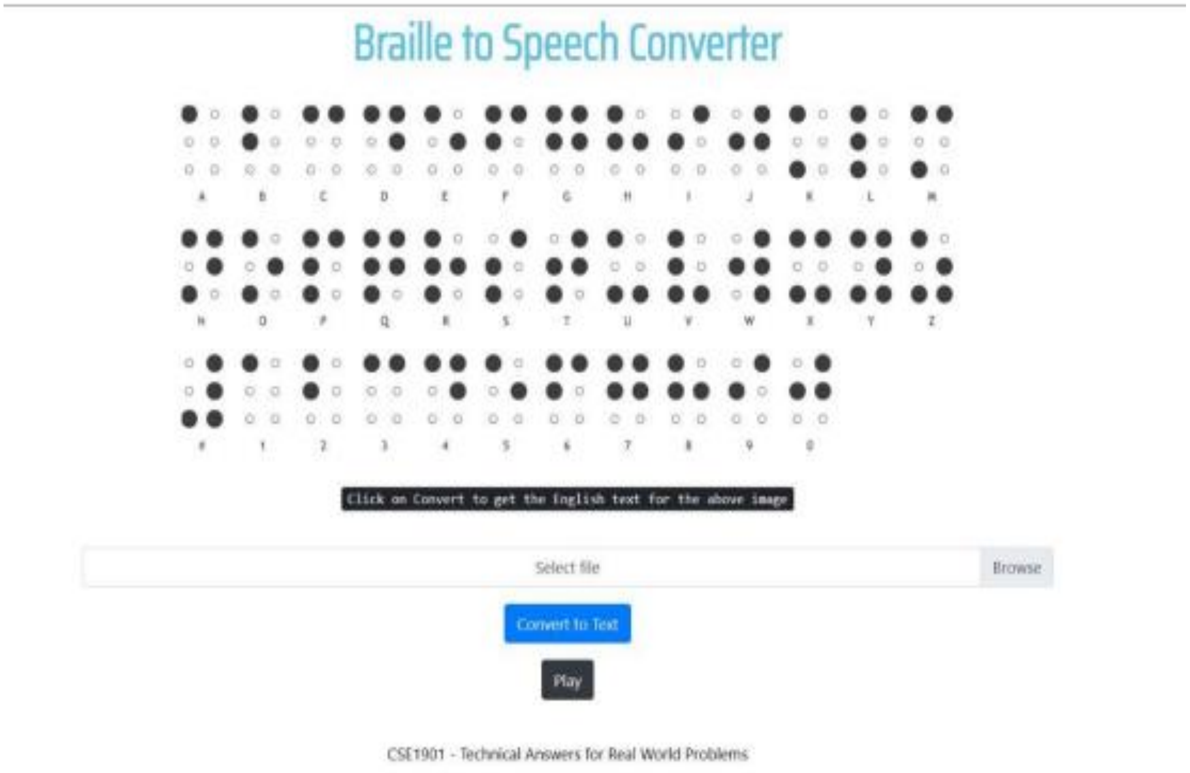
This fox is too quick!

```
*this fox is too quicK*
```

Processed and Digested successfully

CSE1901 - Technical Answers for Real World Problems

5. Output-4

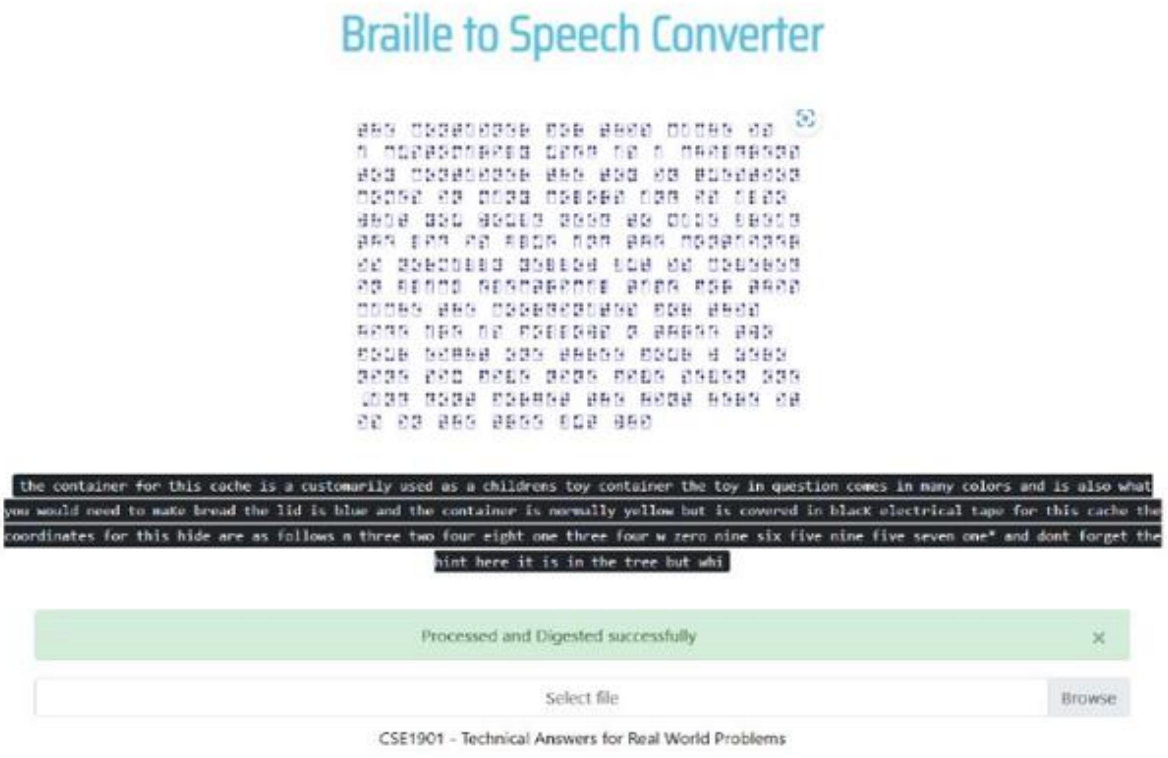


Braille to Speech Converter

Click on Convert to get the English text for the above image

CSE1901 - Technical Answers for Real World Problems

6. Output-5



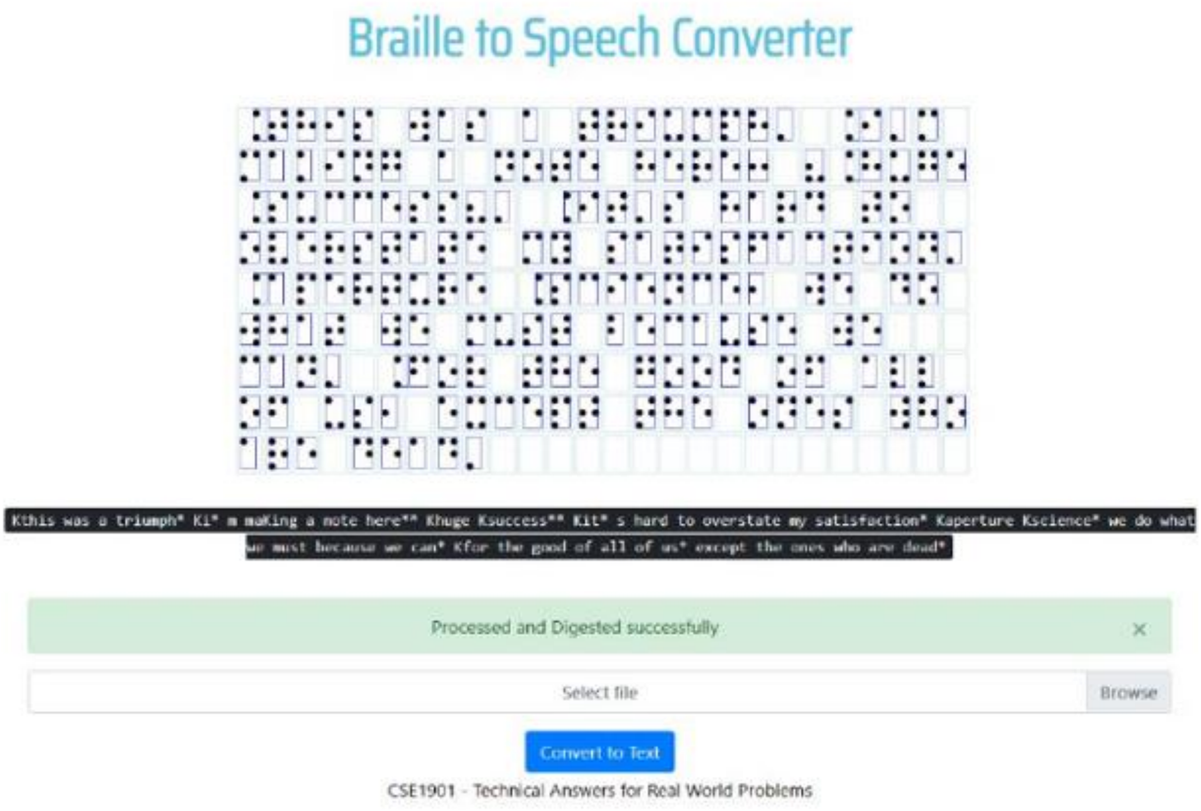
Braille to Speech Converter

the container for this cache is a customarily used as a childrens toy container the toy in question comes in many colors and is also what you would need to make bread the lid is blue and the container is normally yellow but is covered in black electrical tape for this cache the coordinates for this hide are as follows n three two four eight one three four w zero nine six five nine five seven one* and dont forget the hint here it is in the tree but whi

Processed and Digested successfully

CSE1901 - Technical Answers for Real World Problems

7. Output-6


Braille to Speech Converter

```
Kthis was a triumph* Ki* m maKing a note here** Khuge Ksuccess** Kit* s hard to overstate my satisfaction* Kaperture Kscience* we do what
we must because we can* Kfor the good of all of us* except the ones who are dead*
```

Processed and Digested successfully

CSE1901 - Technical Answers for Real World Problems

CODE:

1. **Segmentation Engine**:

```python
import cv2
import numpy as np
from math import sqrt
from collections import Counter
from .BrailleCharacter import BrailleCharacter

class SegmentationEngine(object):
    def __init__(self, image = None):
        self.image = image
        self.initialized = False
        self.dots = []
        self.diameter = 0.0
        self.radius = 0.0
        self.next_epoch = 0
        self.characters = []
        return;

    def __iter__(self):
        return self

    def __next__(self):
        return self.next()

    def next(self):
        if not self.initialized:
            self.initialized = True
            contours = self.__process_contours()
            if len(contours) == 0:
                # Since we have no dots.
                self.__clear()
                raise StopIteration()
            enclosingCircles = self.__get_min_enclosing_circles(contours)
            if len(enclosingCircles) == 0:
                self.__clear()
                raise StopIteration()

            diameter,dots,radius = self.__get_valid_dots(enclosingCircles)
            if len(dots) == 0:
                self.__clear()
                raise StopIteration()
            self.diameter = diameter
            self.dots = dots
            self.radius = radius
            self.next_epoch = 0
            self.characters = []

        if len(self.characters) > 0:
            r = self.characters[0]
            del self.characters[0]
            return r

        cor = self.__get_row_cor(self.dots, epoch=self.next_epoch) # do not respect breakpoints
        if cor is None:
            self.__clear()
            raise StopIteration()

        top = int(cor[1] - int(self.radius*1.5)) # y coordinate
        self.next_epoch = int(cor[1] + self.radius)

        cor = self.__get_row_cor(self.dots,self.next_epoch,self.diameter,True)
        if cor is None:
            # Assume next epoch
            self.next_epoch = int(self.next_epoch + (2*self.diameter))
        else:
            self.next_epoch = int(cor[1] + self.radius)

        cor = self.__get_row_cor(self.dots,self.next_epoch,self.diameter,True)
        if cor is None:
            self.next_epoch = int(self.next_epoch + (2*self.diameter))
        else:
            self.next_epoch = int(cor[1] + self.radius)

        bottom = self.next_epoch
        self.next_epoch += int(2*self.diameter)

        DOI = self.__get_dots_from_region(self.dots, top, bottom)
        xnextEpoch = 0
        while True:
            xcor = self.__get_col_cor(DOI, xnextEpoch)
            if xcor is None:
                break

            left = int(xcor[0] - self.radius) # x coordinate
            xnextEpoch = int(xcor[0] + self.radius)
            xcor = self.__get_col_cor(DOI,xnextEpoch,self.diameter,True)
            if xcor is None:
                # Assumed
                xnextEpoch += int(self.diameter*1.5)
            else:
                xnextEpoch = int(xcor[0]) + int(self.radius)
            right = xnextEpoch
            box = (left, right, top, bottom)
            dts = self.__get_dots_from_box(DOI, box)
            char = BrailleCharacter(dts, self.diameter, self.radius, self.image)
            char.left = left
            char.right = right
            char.top = top
            char.bottom = bottom
            self.characters.append(char)

        if len(self.characters) < 1:
            self.__clear()
            raise StopIteration()

        r = self.characters[0]
        del self.characters[0]
        return r

    def __clear(self):
        self.image = None
        self.initialized = False
        self.dots = []
        self.diameter = 0.0
        self.radius = 0.0
        self.next_epoch = 0
        self.characters = []

    def update(self, image):
        self.__clear()
        self.image = image
        return True

    def __get_row_cor(self, dots, epoch = 0, diameter = 0, respectBreakpoint = False):
        if len(dots) == 0:
            return None
        minDot = None
        for dot in dots:
            x,y = dot[0]
            if y < epoch:
                continue

            if minDot is None:
                minDot = dot
            else:
                v = int(y - epoch)
                minV = int(minDot[0][1] - epoch)
                if minV > v:
                    minDot = dot
                else:
                    continue
        if minDot is None:
            return None
        if respectBreakpoint:
            v = int(minDot[0][1] - epoch)
            if v > (2*diameter):
                return None # indicates that the entire row is not set
        return minDot[0] # (X,Y)

    def __get_col_cor(self, dots, epoch = 0, diameter = 0, respectBreakpoint = False):
        if len(dots) == 0:
            return None
        minDot = None
        for dot in dots:
            x,y = dot[0]
            if x < epoch:
                continue

            if minDot is None:
                minDot = dot
            else:
                v = int(x - epoch)
                minV = int(minDot[0][0] - epoch)
                if minV > v:
                    minDot = dot
                else:
                    continue
        if minDot is None:
            return None
        if respectBreakpoint:
            v = int(minDot[0][0] - epoch)
            if v > (2*diameter):
                return None # indicates that the entire row is not set
        return minDot[0] # (X,Y)

    def __get_dots_from_box(self, dots, box):
        left,right,top,bottom = box
        result = []
        for dot in dots:
            x,y = dot[0]
            if x >= left and x <= right and y >= top and y <= bottom:
                result.append(dot)
        return result

    def __get_dots_from_region(self, dots, y1, y2):
        D = []
        if y2 < y1:
            return D

        for dot in dots:
            x,y = dot[0]
            if y > y1 and y < y2:
                D.append(dot)
        return D

    def __get_valid_dots(self, circles):
        tolerance = 0.45
        radii = []
        consider = []
        bin_img = self.image.get_binary_image()
        for circle in circles:
            x,y = circle[0]
            rad = circle[1]
            # OpenCV uses row major
            # Since we do a bitwise not, white pixels belong to the dot.

            # Go through the x axis and check if all those are white
            # pixels till you reach the rad
            it = 0
            while it < int(rad):
                if bin_img[y,x+it] > 0 and bin_img[y+it,x] > 0:
                    it += 1
                else:
                    break
            else:
                if bin_img[y,x] > 0:
                    consider.append(circle)
                    radii.append(rad)

        baserad = Counter(radii).most_common(1)[0][0]
        dots = []
        for circle in consider:
            x,y = circle[0]
            rad = circle[1]
            if rad <= int(baserad * (1+tolerance)) and rad >= int(baserad * (1-tolerance)):
                dots.append(circle)

        # Remove duplicate enclosing circles
        # (i.e) Remove circle enclosed by another other circle.
        for dot in dots:
            X1,Y1 = dot[0]
            C1 = dot[1]
            for sdot in dots:
                if dot == sdot:
                    continue
                X2,Y2 = sdot[0]
                C2 = sdot[1]
                D = sqrt(((X2 - X1)**2) + ((Y2-Y1)**2))
                if C1 > (D + C2):
                    dots.remove(sdot)

        # Filtered base radius
        radii = []
        for dot in dots:
            rad = dot[1]
            radii.append(rad)
        baserad = Counter(radii).most_common(1)[0][0]
        return 2*(baserad), dots, baserad

    def __get_min_enclosing_circles(self, contours):
        circles = []
        radii = []
        for contour in contours:
            (x,y), radius = cv2.minEnclosingCircle(contour)
            center = (int(x), int(y))
            radius = int(radius)
            radii.append(radius)
            circles.append((center, radius))
        return circles

    def __process_contours(self):
        edg_bin_img = self.image.get_edged_binary_image()
        contours = cv2.findContours(edg_bin_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        if len(contours) == 2:
            contours = contours[0]
        else:
            contours = contours[1]
        return contours

```

2. Braille Classifier

```
from math import sqrt

def get_distance(p1, p2):
        x1,y1 = p1
        x2,y2 = p2
        return ((x2 - x1)**2) + ((y2 - y1)**2)

def get_left_nearest(dots, diameter, left):
        nearest = None
        for dot in dots:
            x,y = dot[0]
            dist = int(x - left)
            if dist <= diameter:
                if nearest is None:
                    nearest = dot
                else:
                    X,Y = nearest[0]
                    DIST = int(X - left)
                    if DIST > dist:
                        nearest = dot
        return nearest

def get_right_nearest(dots, diameter, right):
        nearest = None
        for dot in dots:
            x,y = dot[0]
            dist = int(right - x)
            if dist <= diameter:
                if nearest is None:
                    nearest = dot
                else:
                    X,Y = nearest[0]
                    DIST = int(right - X)
                    if DIST > dist:
                        nearest = dot
        return nearest

def get_dot_nearest(dots, diameter, pt1):
        nearest = None
        diameter **= 2
        for dot in dots:
            point = dot[0]
            dist_from_pt1 = get_distance(point, pt1)
            if dist_from_pt1 <= diameter:
                if nearest is None:
                    nearest = dot
                else:
                    pt = nearest[0]
                    ndist_from_pt1 = get_distance(pt, pt1)
                    if ndist_from_pt1 >= dist_from_pt1:
                        nearest = dot
        return nearest



def get_combination(box, dots, diameter):
        result = [0,0,0,0,0,0]
        left,right,top,bottom = box

        midpointY = int((bottom - top)/2)
        end = (right, midpointY)
        start = (left, midpointY)
        width = int(right - left)

        corners = { (left,top): 1, (right,top): 4, (left, bottom): 3,
(right,bottom): 6,
                (left): 2, (right): 5}

        for corner in corners:
            if corner != left and corner != right:
                D = get_dot_nearest(dots, int(diameter), corner)
            else:
                if corner == left:
                    D = get_left_nearest(dots, int(diameter), left)
                else:
                    D = get_right_nearest(dots, int(diameter), right)

            if D is not None:
                dots.remove(D)
                result[corners[corner]-1] = 1

            if len(dots) == 0:
                break
        return end,start,width,tuple(result);

def translate_to_number(value):
    if value == 'a':
        return '1'
    elif value == 'b':
        return '2'
    elif value == 'c':
        return '3'
    elif value == 'd':
        return '4'
    elif value == 'e':
        return '5'
    elif value == 'f':
        return '6'
    elif value == 'g':
        return '7'
    elif value == 'h':
        return '8'
    elif value == 'i':
        return '9'
    else:
        return '0'

class Symbol(object):
    def __init__(self, value = None, letter = False, special = False):
        self.is_letter = letter
        self.is_special = special
        self.value = value

    def is_valid(self):
        r = True
        r = r and (self.value is not None)
        r = r and (self.is_letter is not None or self.is_special is not None)
        return r

    def letter(self):
        return self.is_letter

    def special(self):
        return self.is_special

class BrailleClassifier(object):
    symbol_table = {
         (1,0,0,0,0,0): Symbol('a',letter=True),
         (1,1,0,0,0,0): Symbol('b',letter=True),
         (1,0,0,1,0,0): Symbol('c',letter=True),
         (1,0,0,1,1,0): Symbol('d',letter=True),
         (1,0,0,0,1,0): Symbol('e',letter=True),
         (1,1,0,1,0,0): Symbol('f',letter=True),
         (1,1,0,1,1,0): Symbol('g',letter=True),
         (1,1,0,0,1,0): Symbol('h',letter=True),
         (0,1,0,1,0,0): Symbol('i',letter=True),
         (0,1,0,1,1,0): Symbol('j',letter=True),
         (1,0,1,0,0,0): Symbol('K',letter=True),
         (1,1,1,0,0,0): Symbol('l',letter=True),
         (1,0,1,1,0,0): Symbol('m',letter=True),
         (1,0,1,1,1,0): Symbol('n',letter=True),
         (1,0,1,0,1,0): Symbol('o',letter=True),
         (1,1,1,1,0,0): Symbol('p',letter=True),
         (1,1,1,1,1,0): Symbol('q',letter=True),
         (1,1,1,0,1,0): Symbol('r',letter=True),
         (0,1,1,1,0,0): Symbol('s',letter=True),
         (0,1,1,1,1,0): Symbol('t',letter=True),
         (1,0,1,0,0,1): Symbol('u',letter=True),
         (1,1,1,0,0,1): Symbol('v',letter=True),
         (0,1,0,1,1,1): Symbol('w',letter=True),
         (1,0,1,1,0,1): Symbol('x',letter=True),
         (1,0,1,1,1,1): Symbol('y',letter=True),
         (1,0,1,0,1,1): Symbol('z',letter=True),
         (0,0,1,1,1,1): Symbol('#',special=True),
    }
    def __init__(self):
        self.result = ''
        self.shift_on = False
        self.prev_end = None
        self.number = False
        return;

    def push(self, character):
        if not character.is_valid():
            return;
        box = character.get_bounding_box()
        dots = character.get_dot_coordinates()
        diameter = character.get_dot_diameter()
        end,start,width,combination = get_combination(box, dots, diameter)

        if combination not in self.symbol_table:
            self.result += "*"
            return;

        if self.prev_end is not None:
            dist = get_distance(self.prev_end, start)
            if dist*0.5 > (width**2):
                self.result += " "
        self.prev_end = end

        symbol = self.symbol_table[combination]
        if symbol.letter() and self.number:
            self.number = False
            self.result += translate_to_number(symbol.value)
        elif symbol.letter():
            if self.shift_on:
                self.result += symbol.value.upper()
            else:
                self.result += symbol.value
        else:
            if symbol.value == '#':
                self.number = True
        return;

    def digest(self):
        return self.result

    def clear(self):
        self.result = ''
        self.shift_on = False
        self.prev_end = None
        self.number = False
        return;
```

3. Braille Image

```
import cv2
import numpy as np

class BrailleImage(object):
    def __init__(self, image):
        # Read source image
        self.original = cv2.imread(image)
        if self.original is None:
            raise IOError('Cannot open given image')

        # First Layer, Convert BGR(Blue Green Red Scale) to Gray Scale
        gray = cv2.cvtColor(self.original, cv2.COLOR_BGR2GRAY)

        # Save the binary image of the edge detected
        self.edged_binary_image = self.__get_edged_binary_image(gray)

        # Now do the same to save a binary image to get the contents
        # inside the edges to see if the dot is really filled.
        self.binary_image = self.__get_binary_image(gray)
        self.final = self.original.copy()
        self.height, self.width, self.channels = self.original.shape
        return;

    def bound_box(self,left,right,top,bottom,color= (255,0,0), size=1):
        self.final = cv2.rectangle(self.final, (left, top), (right, bottom),
color, size)
        return True

    def get_final_image(self):
        return self.final

    def get_original_image(self):
```

```python
        return self.original

    def get_edged_binary_image(self):
        return self.edged_binary_image

    def get_binary_image(self):
        return self.binary_image

    def get_height(self):
        return self.height

    def get_width(self):
        return self.width

    def __get_edged_binary_image(self, gray):
        # First Lvl Blur to Reduce Noise
        blur = cv2.GaussianBlur(gray,(3,3),0)
        # Adaptive Thresholding to define the  dots in Braille
        thres = cv2.adaptiveThreshold(
                blur,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,5,4)
        # Remove more Noise from the edges.
        blur2 = cv2.medianBlur(thres,3)
        # Sharpen again.
        ret2,th2 = 
cv2.threshold(blur2,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
        # Remove more Noise.
        blur3 = cv2.GaussianBlur(th2,(3,3),0)
        # Final threshold
        ret3,th3 = 
cv2.threshold(blur3,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
        return cv2.bitwise_not(th3)

    def __get_binary_image(self, gray):
        blur      = cv2.GaussianBlur(gray,(3,3),0)
        ret2,th2 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
        blur2     = cv2.medianBlur(th2,3)
        ret3,th3 = 
cv2.threshold(blur2,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
        return cv2.bitwise_not(th3)
```

4. Braille Character

```python
class BrailleCharacter(object):
    def __init__(self, dot_coordinates, diameter, radius, parent_image):
        self.left = None
        self.right = None
        self.top = None
        self.bottom = None
        self.dot_coordinates = dot_coordinates
        self.diameter = diameter
        self.radius = radius
        self.parent_image = parent_image
        return;

    def mark(self):
        self.parent_image.bound_box(self.left,self.right,self.top,self.bottom)
        return;

    def get_parent_image(self):
        return self.parent_image

    def get_dot_diameter(self):
        return self.diameter

    def get_dot_radius(self):
        return self.radius

    def get_dot_coordinates(self):
        return self.dot_coordinates

    def get_left(self):
        return self.left

    def get_right(self):
        return self.right

    def get_top(self):
        return self.top

    def get_bottom(self):
        return self.bottom

    def get_opencv_left_top(self):
        return (self.left, self.top)

    def get_opencv_right_bottom(self):
        return (self.right, self.bottom)

    def get_bounding_box(self, form = "left,right,top,bottom"):
        r = []
        form = form.split(',')
        if len(form) < 4:
            return (self.left,self.right,self.top,self.bottom)

        for direction in form:
            direction = direction.lower()
            if direction == 'left':
                r.append(self.left)
            elif direction == 'right':
                r.append(self.right)
            elif direction == 'top':
                r.append(self.top)
            elif direction == 'bottom':
                r.append(self.bottom)
            else:
                return (self.left,self.right,self.top,self.bottom)

        return tuple(r)

    def is_valid(self):
        r = True
        r = r and (self.left is not None)
        r = r and (self.right is not None)
        r = r and (self.top is not None)
        r = r and (self.bottom is not None)
        return r
```

CONCLUSION

In conclusion, the Braille to speech converter is a vital assistive technology that enables individuals with visual impairments to access written information more independently and effectively. The development of a Braille to speech converter requires careful consideration of hardware and software requirements, including the parsing algorithm, text-to-speech engine, language processing, user interface, customization options, error handling, compatibility, documentation, testing, security, and privacy.

Through the implementation of the Braille to speech converter, individuals with visual impairments can have access to a wide range of information, including books, articles, documents, and other written content, in a format that is accessible to them through speech output. This technology has the potential to significantly enhance the independence, education, and overall quality of life of individuals with visual impairments, enabling them to engage in educational, professional, and recreational activities more effectively.

However, it is important to acknowledge that the Braille to speech converter may have limitations and challenges, such as accuracy and reliability of Braille parsing, voice quality of the TTS engine, and compatibility with different Braille standards and file formats. Therefore, continuous improvement, user feedback, and iterative updates to the system should be prioritized to ensure that it meets the evolving needs and preferences of users with visual impairments.

In conclusion, the Braille to speech converter is a valuable technology that has the potential to greatly benefit individuals with visual impairments by providing them with access to written information. Further research, development, and refinement of the Braille to speech converter can contribute to the advancement of accessibility and inclusion for individuals with visual impairments, promoting equal opportunities and empowerment.

FUTURE WORK

The Braille to Speech Converter is an innovative assistive technology that aims to improve accessibility for individuals with visual impairments. As we look towards the future, there are several potential areas of further work and development to enhance the functionality and usability of this technology. Here are some potential suggestions for a Future Work section in a report on the topic:

1. Enhanced Accuracy and Speed: As with any speech recognition technology, improving accuracy and speed would be a top priority. Advancements in natural language processing (NLP) and machine learning algorithms could be employed to fine-tune the Braille to Speech Converter's recognition capabilities. This could involve further training the system on diverse Braille inputs and optimizing the conversion process to generate more accurate and fluent speech output in real-time.

2. Expanded Language Support: Currently, the Braille to Speech Converter may be limited to a specific language or a set of predefined vocabulary. Expanding language support to include multiple languages, and incorporating domain-specific terminologies, technical jargon, and slang would greatly enhance its usability and versatility for a wider range of users.

3. Integration with Other Devices and Platforms: Integrating the Braille to Speech Converter with other devices and platforms could further extend its accessibility and usability. For example, integration with smartphones, smart speakers, or wearable devices could allow users to access the converter on-the-go, making it more convenient and user-friendly.

4. User Interface and Accessibility Improvements: The user interface of the Braille to Speech Converter could be improved to make it more intuitive, easy-to-use, and accessible for individuals with visual impairments. Incorporating features such as voice-guided instructions, haptic feedback, and customizable settings could enhance the overall user experience and make the technology more user-friendly.

5. User Feedback and Testing: Gathering feedback from users with visual impairments and incorporating their input in the ongoing development process could be invaluable in improving the Braille to Speech Converter. Conducting user testing sessions and usability studies to identify any challenges or limitations in real-world scenarios and addressing them through iterative design and development cycles could lead to significant enhancements in the technology's effectiveness and usability.

6. Cost Reduction and Scalability: Exploring ways to reduce the cost of production, implementation, and maintenance of the Braille to Speech Converter could make it more affordable and accessible to a wider range of users. Scalability considerations, such as developing versions of the technology for different Braille systems or form factors, could also be explored to accommodate diverse user needs and preferences.

7. Real-world Implementation and Adoption: Finally, focusing on real-world implementation and adoption of the Braille to Speech Converter could be a critical aspect of future work. Collaborating with stakeholders, such as advocacy groups, educational institutions, and assistive technology providers, to promote awareness, adoption, and integration of the technology into daily life could lead to meaningful impact and positive change for individuals with visual impairments.

In conclusion, the Braille to Speech Converter has the potential for further development and improvements to enhance its accuracy, language support, usability, and real-world adoption. Through continued research, development, and collaboration with relevant stakeholders, this technology has the potential to significantly improve the accessibility and inclusivity for individuals with visual impairments and pave the way for a more inclusive future.

REFERENCES

1. https://ieeexplore.ieee.org/document/8282637

2. https://scholar.ppu.edu/handle/123456789/658

3. http://www.kscst.iisc.ernet.in/spp/41_series/SPP41S/01_Seminar_Projects/083_41S_BE_1713.pdf

4. https://www.brailletranslator.org/

5. https://www.researchgate.net/publication/342169863_Speech_to_Braille_Convertor_for_Visually_Impaired_Using_Python

6. https://link.springer.com/chapter/10.1007/978-1-84628-867-8_14

7. https://www.semanticscholar.org/paper/Braille-Converter-and-Text-To-Speech-Translator-for-Silva-Wedasinghe/296172844cfff0c690ded435c4b05afc8df59bf5

8. https://www.ijser.org/paper/FPGA-Based-Braille-to-Text-Speech-For-Blind-Persons.html

9. https://d-nb.info/1257394266/34

10. https://bvmengineering.ac.in/NAAC/Criteria1/1.3/1.3.4/Micrphub_Faizan.pdf