BRAILLE TRANSLATOR WITH PYTHON AND DEEP LEARNING

by Harshith Gudladhana

CAPSTONE PROJECT REPORT (Applicable to CSE/IT students only)

CONTENTS OF THE REPORT

- Cover page   – on hard paper
- Inner page   – same as cover page but on the soft paper
- PAC form
- Declaration
- Certificate
- Acknowledgement
- Table of Contents

1. Introduction
2. Profile of the Problem. Rationale/Scope of the study (Problem Statement)
3. Existing System
   - Introduction
   - Existing Software
   - DFD for present system
   - What's new in the system to be developed
4. Problem Analysis
   - Product definition
   - Feasibility Analysis
   - Project Plan
5. Software Requirement Analysis
   - Introduction
   - General Description
   - Specific Requirements
6. Design
   - System Design
   - Design Notations
   - Detailed Design
   - Flowcharts
   - Pseudo code

7. Testing
- Functional testing
- Structural testing
- Levels of testing
- Testing the project

8. Implementation
- Implementation of the project
- Conversion Plan
- Post-Implementation and Software Maintenance

9. Project Legacy
- Current Status of the project
- Remaining Areas of concern
- Technical and Managerial lessons learnt

10. User Manual: A complete document (Help Guide) of the software developed.

11. Source Code (where ever applicable) or System Snapshots

12. Bibliography

CAPSTONE PROJECT REPORT

(Project Term January-May 2023)

BRAILLE TRANSLATOR WITH PYTHON AND DEEP LEARNING

Submitted by

(Gudladhana Harshith)   Registration Number :11903248
(Bokam Kusuma)   Registration Number :11905583
(Jeevnesh Krishan Vyas)   Registration Number :11911089
(Saurav Kumar)   Registration Number :11903912

Project Group Number: KC416

Course Code: CSE445

Under the Guidance of

(Jaspreet Kaur, Assistant Professor)

School of Computer Science and Engineering

PAC Form

LOVELY PROFESSIONAL UNIVERSITY

TOPIC APPROVAL PERFORMA

School of Computer Science and Engineering (SCSE)

Program : P132-H: B.Tech. (Computer Science & Engineering) (Hons.)

COURSE CODE : CSE445   **REGULAR/BACKLOG :** Regular   **GROUP NUMBER :** CSEPGC0416

Supervisor Name : Jaspreet Kaur   **UID :** 29511   **Designation :** Assistant Professor

Qualification : _____   **Research Experience :** _____

SR.NO.	NAME OF STUDENT	Prev. Regd. No.	BATCH	SECTION	CONTACT NUMBER
1	Jeevnesh Krishan Vyas	11913089	2019	K19PT	8580740779
2	Roushan Kumar	11906794	2019	K19FR	7481831758
3	K Akshith	11906654	2019	K19AR	9515400291
4	Bokam Kusuma	11905583	2019	K19PV	9952113771
5	Saurav kumar	11913912	2019	K19PG	6299387644
6	Gudladhana Harshith	11903248	2019	K19YG	8897138835

SPECIALIZATION AREA : Programming-I   **Supervisor Signature :** _____

PROPOSED TOPIC :   Braille Translator (Braille Text to Speech) using Python and Deep Learning algorithms.

	Qualitative Assessment of Proposed Topic by PAC	
Sr.No.	**Parameter**	**Rating (out of 10)**
1	Project Novelty: Potential of the project to create new knowledge	7.75
2	Project Feasibility: Project can be timely carried out in-house with low-cost and available resources in the University by the students.	7.75
3	Project Academic Inputs: Project topic is relevant and makes extensive use of academic inputs in UG program and serves as a culminating effort for core study area of the degree program.	7.00
4	Project Supervision: Project supervisor's is technically competent to guide students, resolve any issues, and impart necessary skills.	7.75
5	Social Applicability: Project work intends to solve a practical problem.	7.00
6	Future Scope: Project has potential to become basis of future research work, publication or patent.	7.00

PAC Committee Members		
PAC Member (HOD/Chairperson) Name: Dr. Vijay Kumar Garg	UID: 14085	Recommended (Y/N): Yes
PAC Member (Allied) Name: Ravi Kant Sahu	UID: 16920	Recommended (Y/N): NA
PAC Member 3 Name: Virrat Devaser	UID: 14591	Recommended (Y/N): Yes

Final Topic Approved by PAC: Braille Translator (Braille Text to Speech) using Python and Deep Learning algorithms.

Overall Remarks: Approved

PAC CHAIRPERSON Name: 25708: Dr. Rachit Garg   **Approval Date:**   11 Apr 2023

5/5/2023 11:42:48 AM

ii

DECLARATION

We hereby declare that the project work entitled ("Title of the project") is an authentic record of our own work carried out as requirements of Capstone Project for the award of B.Tech degree in _____(Programme Name) from Lovely Professional University, Phagwara, under the guidance of Jaspreet Kaur, during January to May 2023. All the information furnished in this capstone project report is based on our own intensive work and is genuine.

Project Group Number: ....

Name of Student 1: Gudladhana Harshith
Registration Number:11903248

Name of Student 2: Bokam Kusuma
Registration Number: 11905583

Name of Student 3: Jeevnesh Krishan Vyas
Registration Number: 11911089

Name of Student 4: Saurav Kumar
Registration Number: 11903912

Gudladhana Harshith
Date:

Bokam Kusuma
Date:

Jeevnesh Krishan Vyas
Date:

Saurav Kumar
Date:

CERTIFICATE

This is to certify that the declaration statement made by this group of students is correct to the best of my knowledge and belief. They have completed this Capstone Project under my guidance and supervision. The present work is the result of their original investigation, effort and study. No part of the work has ever been submitted for any other degree at any University. The Capstone Project is fit for the submission and partial fulfillment of the conditions for the award of B.Tech degree in _____ (Programme Name) from Lovely Professional University, Phagwara.

Signature and Name of the Mentor

Assistant Professor

School of Computer Science and Engineering.
Lovely Professional University,
Phagwara, Punjab.

Date:

ACKNOWLEDGEMENT

Acknowledgement Form for Student Project

Project Title: [Title of the Project]

I, [Name of the Student], a student of [Class/Year/Program] of [School/College/University], hereby acknowledge that I have completed the project titled [Title of the Project] under the guidance of [Name of the Guide/Supervisor/Teacher].

I confirm that I have conducted this project ethically and have given due credit to all the sources of information and support that I have used. I understand that any act of plagiarism, misrepresentation of facts, or unethical practices can lead to serious consequences, including disqualification of my project.

I also acknowledge the support and guidance provided by [Name(s) of any other person(s) who provided assistance]. I confirm that their contributions have been duly recognized in the project report and any other related documents.

I understand that this acknowledgement form serves as proof of my completion of the project and that it will be kept as a record by the school/college/university. I also understand that the project report and related materials may be used for academic and research purposes, with due credit to the author.

Signature: _____

Date: _____

TABLE OF CONTENTS

Profile of the Problem. Rationale/Scope of the study (Problem Statement)

Existing System

- Introduction
- Existing Software
- DFD for present system
- What's new in the system to be developed

Problem Analysis

- Product definition
- Feasibility Analysis
- Project Plan

Software Requirement Analysis

- Introduction
- General Description
- Specific Requirements

6. Design

- System Design
- Design Notations
- Detailed Design
- Flowcharts
- Pseudo code

7. Testing

- Functional testing
- Structural testing

1. INTRODUCTION

Braille is a writing and reading method used by blind or low-vision persons. It was invented in the nineteenth century by Louis Braille (his picture is displayed in figure 1), a blind Frenchman. Braille represents letters, numerals, punctuation marks, and other symbols with a series of raised dots organised in various patterns.

The braille system's fundamental unit is a cell, which has six dots grouped in two columns of three dots each. Different characters and symbols can be expressed by altering the pattern of raised dots within the cell. For example, the letter "A" is represented by a single dot in the upper left corner of the cell, but the letter "B" is represented by two dots.

Braille, in addition to the fundamental letters of the alphabet, adds contractions and abbreviated forms that allow for more efficient writing. For example, the word "and" is represented in the cell by a single dot, but the word "but" is represented by a dot in the cell's lower left corner. Reading braille entails moving one's fingertips over raised dots on paper or a braille display.

Many blind individuals learn to read and write braille from an early age, and it is frequently used in schools, libraries, and other places to allow persons who are blind or have impaired vision with access to written information.

Figure 1: Potrait of Louis Braille[13]

1.1 GRADE 1 BRAILLE

Braille Grade 1 is the most basic type of braille writing. It is also known as uncontracted braille or Grade 1 braille. It includes the fundamental braille alphabet, punctuation marks, and some basic formatting options. Each letter of the alphabet in Grade 1 braille is represented by its own unique combination of dots within a braille cell. The basic braille alphabet is made up of 26 letters represented by six dots grouped in two columns of three dots each.

For example, the letter "A" is represented by a single dot in the cell's upper left corner, but the letter "B" is represented by dots in the cell's upper left and middle left corners. In Grade 1 braille, punctuation symbols include the period, Among these are the comma, colon, semicolon, question mark, and exclamation point. Furthermore, formatting elements such as line breaks and paragraph breaks are represented in Grade 1 braille to aid in indicating the structure of written text.

Grade 1 braille is the most popular kind of braille learned by blind or visually impaired people, and it is often used for marking goods such as household items, files, and books. However, because to its restricted scope, reading lengthy or complicated texts printed in Grade 1 braille might be challenging. As

a result, many persons who use braille eventually progress to Grade 2 braille, which contains a broader range of contractions and abbreviations to facilitate reading and writing.

Uncontracted (Grade 1) Braille									
a	b	c	d	e	f	g	h	i	j
k	l	m	n	o	p	q	r	s	t
u	v	x	y	z	w				

Figure 2 :Figure showing the alphabetical representation of braille characters

1.2 Grade 2 Braille

Braille Grade 2 is a more sophisticated style of braille writing that contains more contractions and abbreviations to make reading and writing more efficient. It is also known as contracted braille or Grade 2 braille. Grade 2 braille represents commonly used words and letter combinations with a single braille character or combination of characters, similar to how shorthand is used in standard written text.

The Grade 2 braille system has around 180 contractions, which are braille cell combinations that represent typical words or phrases. The word "and", for example, is represented by a single cell with dots in the middle and bottom rows, but the word "with" is represented by a single cell with dots in the middle and top rows.

Grade 2 braille contains short forms, which are shorter versions of commonly used words, in addition to contractions. For example, the word "before" is represented by two characters ("be") followed by a contraction for "fore."

Learning Grade 2 braille takes more time and work than learning Grade 1 braille, but once mastered, it allows for speedier reading and writing. It is the most common type of braille used for lengthier materials like books, periodicals, and newspapers. Furthermore, many current braille displays and electronic devices enable Grade 2 braille input and output, making it a necessary ability for those who rely on braille for reading and writing.
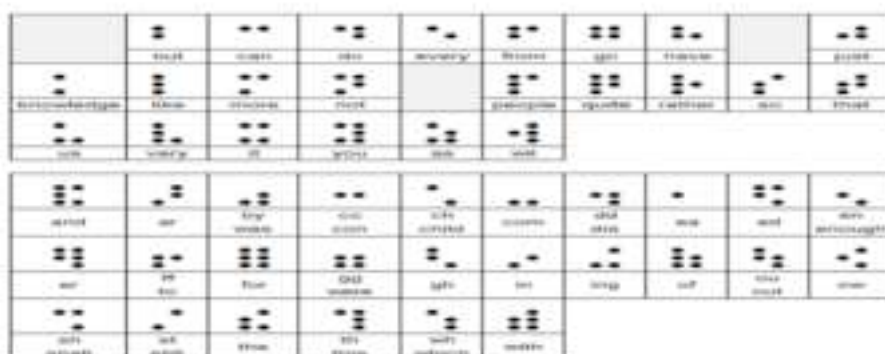
Figure 3: Figure showing the Grade 2 Braille Representation

2.Scope of the study

This study tackles the requirement for an efficient and accurate braille translation system capable of converting braille text to readable text and text-to-speech synthesis. Flask is used as the web framework, OpenCV for image processing, Optical Braille recognition to recognize the braille letters in the picture, and a web synthesizer to transform the translated text to audio. Many blind and visually impaired people now experience difficulties reading braille texts due to the scarcity of braille materials and the expensive cost of braille translation machines.

As a result, there is a need for a low-cost, efficient, and accurate braille translation system that those with visual impairments may readily access and utilise.

The suggested system tackles this issue by offering an accurate and efficient way of converting braille text into readable text, as well as text-to-speech synthesis.

The system processes the picture of the braille text with OpenCV and recognises the braille letters in the image with Optical Braille recognition. Flask is used to create an online application that allows users to submit an image of braille text, after which the system converts the braille text to readable language and outputs the translated text as speech using a web synthesiser.

The study's goal is to create and assess the accuracy and efficiency of the proposed braille translation system. The study's goal is to examine the system's performance in converting braille text to readable text as well as the effectiveness of the text-to-speech synthesis capability. The study will also analyse the online application's usability and accessibility, as well as the system's potential advantages in boosting access to braille publications for those with visual impairments.

3.Existing System:

3.1 Introduction:

The current braille translation system is based on classic braille translation machines, which are costly and require specialised training to use. These devices often include a braille display that allows users to view braille text as well as a braille keyboard that allows users to enter braille text.

However, the expensive expense of these gadgets prevents many people with visual impairments from using them, particularly in low-income nations. Furthermore, the current system has limited capabilities

and functionality. Traditional braille translation devices do not include text-to-speech synthesis, limiting their use for people who cannot read braille or want to listen to material rather than read it in braille.

3.2 Existing Software:

There are now various software solutions available for braille translation. Among the most widely utilised systems are:

1. Duxbury Braille Translator: Duxbury is a software tool that can convert braille text into print, audio, and electronic forms. It can produce braille textbooks, manuals, and other documents in both contracted and uncontracted braille and is widely used by schools and organizations.

2. BrailleBlaster: The American Printing House for the Blind created BrailleBlaster, a free, open-source braille translation programme. It can create braille documents in both contracted and uncontracted braille and provides a variety of formatting choices.
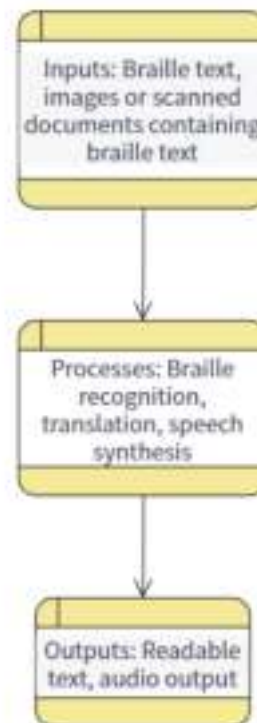
Figure 5: DFD of the Existing System

3.4 What is new to be developed:

1. Web-based interface: The system is created with Flask, a web framework that allows users to access the translation system via a web browser rather than installing software on their devices.

4

2. Optical Braille recognition: The system employs Optical Braille recognition, a cutting-edge image processing approach that recognises the braille letters in the picture, enhancing braille recognition accuracy.

3. Text-to-speech synthesis: The system contains a web synthesiser that turns translated text into voice, allowing visually impaired people to access the translated material through audio output.

4. OpenCV integration: The system uses OpenCV, an open-source computer vision library, to process the picture of the braille text and extract the braille letters, therefore improving the accuracy of the braille recognition process.

Overall, the system to be built in braille translators intends to give those with visual impairments with a quick, accurate, and accessible means of translating braille material into understandable text and audio output. The system has new features that increase braille identification accuracy, a web-based interface, and text-to-speech synthesis, making it a vital tool for improving access to braille content.

4.Product Analysis:

Braille Translator System Product Definition OpenCV, Optical Braille Recognition (OBR), Web Synthesiser, and Flask were used to create this project. Braille Translator System is the name of the product. Product Information:

The Braille Translator System is a cutting-edge solution that uses cutting-edge technologies such as OpenCV, Optical Braille Recognition (OBR), Web Synthesiser, and Flask to provide a fast, accurate, and accessible method of translating braille text into readable text and audio output for people with visual impairments.

4.1.1 Product Features:

1. Optical Braille Recognition (OBR): The system employs OBR to recognize braille letters in images, boosting braille recognition accuracy.

2. OpenCV Integration: The system uses OpenCV, an open-source computer vision toolkit, to analyses the braille text picture and extract the braille letters, improving the accuracy of the braille recognition process.

3. Web-based Interface: The system is created with Flask, a web framework that allows users to access the translation system via a web browser rather than installing software on their devices. Translation of Braille Text: The system translates braille text into readable text using a database of braille characters and contraction rules.

4. Text-to-voice Synthesis: The system contains a web synthesizer that turns translated text into voice, allowing visually impaired people to access the translated material via audio output.

5. User-Friendly Interface: The system has a simple and straightforward user interface that allows users to submit braille text pictures, examine translated text, and listen to speech output.

6. Target Audience: The Braille Translator System's target audience is visually impaired people who use braille as their major mode of communication and require an accessible and accurate way of translating braille text into legible text and audio output.

7. The method can also benefit educators, carers, and organisations that assist visually impaired people.

4.1.2 Benefits:

1. Improved Accessibility: The Braille Translator System provides a more accessible means of translating braille content into understandable text and audio output, allowing visually challenged users easier access to braille information.

2. Higher Accuracy: The system employs OBR and OpenCV to increase braille recognition accuracy, resulting in more accurate translation of braille text.

3. User-Friendly Interface: The system has a simple and intuitive user interface that makes it easy for visually impaired people and their carers to use.

4. Low-Cost: The Braille Translator System is a low-cost solution that does not require any specialized gear or software, allowing it to be used by a broader audience.

5. Technical Specifications: OpenCV for image processing and braille character extraction. Optical Braille Recognition (OBR) is used to identify the braille characters in an image.

5 Software Requirement Analysis:

Software Requirement Analysis of Braille Translator System Developed with OpenCV, Optical Braille Recognition (OBR), Web Synthesizer, and Flask

The Braille Translator System is an innovative solution that leverages the latest technologies to provide an efficient, accurate, and accessible method of translating braille text into readable text and audio output for individuals with visual impairments. The software requirements for the Braille Translator System can be divided into several categories:

1. Functional Requirements: picture Processing: Using OpenCV, the system should be able to process the picture of the braille text and properly extract the braille letters.

2. Optical Braille Recognition (OBR): Using OBR, the system should be able to recognise the braille characters in the image, improving the accuracy of braille recognition.

3. Translation: Using a database of braille characters and contraction rules, the system should be able to convert recognised braille letters into legible text.

4. Text-to-Speech Synthesis: The system should be able to convert the translated text into speech output using a web synthesiser.

5. User Interface: The system should have an easy-to-use interface that allows users to submit braille text pictures, see translated text, and listen to voice output.

6. Requirements for Performance: precision: The system should recognise and translate braille characters with high precision.

7. The system should be fast in processing photos and translating braille text. Scalability: The system should be capable of handling several user requests at the same time.

5.1 Usability Requirements:

1. Accessibility: The system should be usable by visually challenged people and have a user-friendly interface.

2. Compatibility: The system should work with a variety of web browsers and operating systems.

3. Learnability: The system should be simple to understand and use for visually impaired people and their careers.

The Braille Translator System is a complicated software application that relies on a number of critical features to ensure accuracy, speed, usability, and security. The system should be developed with the user in mind, with a simple and intuitive interface that is accessible to people who are visually impaired. The system should also be built to manage several user requests at the same time, as well as be expandable to meet the demands of an expanding user population. Finally, the system should be frequently updated and have a support team to assist users with any problems they may have while using the system.

5.2 GENERAL DESCRIPTION:

The Braille Translator System is a software programme that converts braille text into understandable text and audio output. OpenCV, Optical Braille Recognition (OBR), Web Synthesizer, and Flask are among the technologies used to create the system.

OpenCV is utilized for image processing, allowing the system to process pictures of braille text and reliably extract the braille letters. Optical Braille Recognition (OBR) is used to recognize braille characters in images, which enhances braille recognition accuracy. Using a database of braille characters and contraction rules, the system then converts the recognized braille characters into readable text. Once the text has been translated, the system employs a web synthesizer to transform it into visually accessible voice output.

The system has an easy-to-use interface that allows users to submit braille text pictures, examine translated text, and listen to speech output. The web application is built with Flask, which provides a scalable and secure method for managing many user requests at the same time.

5.3 SPECIFIC REQIREMENTS:

To work effectively, the Braille Translator system requires various dependencies. OpenCV-Python-Headless, Flask, Requests, Requests-Cache, Werkzeug, and Gunicorn are among the requirements.

For image processing, OpenCV-Python-Headless is employed, allowing the system to reliably extract braille letters from images. Flask is used to create the web application, which provides a scalable and secure method for managing many user requests at the same time. Requests is a Python module for making HTTP requests, which may be used to get and deliver data between the server and the client.

Requests-Cache is a caching library that can assist increase system performance and efficiency by storing replies from web requests.

Werkzeug is a Python framework for developing web applications that includes a set of utilities for managing HTTP requests and answers.

Finally, Gunicorn is a Python WSGI HTTP Server that is used for web application deployment, offering a dependable and efficient solution for handling HTTP requests. Overall, the combination of these dependencies enables the Braille Translator system to provide visually impaired users with an accurate, efficient, and accessible solution for translating braille text into readable text and audio output.

6.DESIGN

6.1 SYSTEM DESIGN:

The Braille Translator system design is comprised of numerous components that work together to give an efficient and accurate solution for translating braille text into understandable text and audio output. These components include OpenCV, Optical Braille Recognition (OBR), Web Synthesizer, and Flask.

OpenCV:

The open-source, cross-platform OpenCV (Open-Source Computer Vision Library) library is extensively utilized in computer vision and machine learning applications. Intel initially developed OpenCV in 1999, and in 2000, it was made available as an open-source library. Since then, it has grown to be one of the most widely used libraries for image processing and computer vision.

OpenCV is written in C++ and upholds different programming dialects including Python, Java, and MATLAB. It has over 2500 optimized algorithms for face detection, object detection, image recognition,

tracking, and other computer vision tasks. Image filtering, transformation, feature detection, and optical flow estimation are just a few of the many image and video processing tools offered by OpenCV.

One of the critical elements of OpenCV is its capacity to work progressively with cameras and video transfers. It upholds an extensive variety of camera models and video input designs, making it an integral asset for growing ongoing PC vision applications like expanded reality, mechanical technology, and independent vehicles.

32 Neural networks, decision trees, and support vector machines are just a few of the machine learning algorithms that are supported by OpenCV. Machine learning models for computer vision tasks like object detection and recognition can now be created thanks to this.

OpenCV has an enormous and dynamic local area of designers, specialists, and fans who add to its turn of events and upkeep. This community offers extensive documentation, tutorials, and sample code to help new developers.

OpenCV has been utilized in a large number of utilizations, including picture and video altering, mechanical technology, independent vehicles, security and observation, and clinical imaging. It has also been utilized in industrial applications like quality control and automation, as well as in research projects like the creation of computer vision models for environmental monitoring.

OpenCV is a comprehensive set of tools for computer vision and image processing tasks that is both powerful and adaptable. OpenCV is a useful tool for anyone working in computer vision and machine learning. It has a lot of features, is easy to use, and has a lot of support from the community.

The first component is an image processing module built with OpenCV. This module properly extracts individual braille characters from the picture of the braille text.

```
cv2.imwrite('{}/{}-proc.png'.format(tempdir.name,filename), img.get_final_image())
```

Figure 10: image showing code that uses openCV

Optical Braille Recognition:

Computers can now recognize and interpret Braille text from images or scans thanks to Optical Braille Recognition (OBR). OBR is a significant instrument for outwardly weakened people, as it permits them to change over printed or manually written text into Braille that can be perused utilizing a Braille show or printer.

The course of OBR includes a few stages. A scanner or camera is used to first take a picture of the text, which can be printed or written on paper. The picture is then preprocessed to eliminate clamor and upgrade the differentiation and clearness of the text. This is done to guarantee that the computer can accurately recognize the Braille dots and that they are easily visible.

The image is then broken up into individual characters or characters in groups. This is finished by recognizing the area of the Braille dabs in the picture and gathering them in light of their situation. When the characters have been divided, they are perceived utilizing a blend of example acknowledgment and AI calculations.

The example acknowledgment calculations are intended to recognize the particular examples of raised dabs that make up each Braille character. This is finished by dissecting the spatial connections between the specks and recognizing the particular setup of dabs that relates to each person.

The recognition process's accuracy is improved over time with the help of machine learning algorithms. These calculations utilize measurable procedures to gain from past acknowledgment results and change their acknowledgment models likewise. The system can thus adjust to new Braille characters or writing styles and improve its accuracy.

After the characters are identified, they are combined into words and sentences that can be output as Braille text. The Braille text can be read with the fingertips of visually impaired individuals by being displayed on a Braille display or printed using a Braille printer.

In general, optical Braille recognition is a crucial technology that makes it possible for people with visual impairments to read printed or handwritten text. OBR systems are able to precisely recognize and interpret Braille text from images or scans by utilizing a combination of image processing, pattern recognition, and machine learning techniques. As a result, they serve as an important tool for accessibility and inclusivity.

The Optical Braille Recognition (OBR) module, which recognizes the individual braille characters in the picture, is the second component. This enhances the accuracy of braille recognition, ensuring that the system accurately interprets braille text.

Figure 11: Snapshot of code showing the module using Optical Braille Recognition (OBR)

Translation Module:

The third component is the translation module, which employs a database of braille characters and contraction rules to facilitate translation.

The interpretation module in a Braille interpreter is a urgent part that empowers clients to change over text into Braille. The translation of text into the Braille code, which is a system of raised dots arranged in a particular pattern to represent letters, numbers, and punctuation marks, is the responsibility of this module.

10

A Braille translator's translation module typically converts the user's input text into Braille code according to predetermined rules. These principles direct how each letter, number, and accentuation imprint ought to be addressed in Braille, in view of the standard Braille code.

It is essential to take into consideration the various nuances of the Braille code in order to construct a translation module that is both reliable and accurate. For instance, the Braille code has distinct codes for symbols and punctuation marks in addition to distinct representations for uppercase and lowercase letters.

To deal with these intricacies, the interpretation module commonly utilizes query tables or calculations that map each info character to its relating Braille portrayal. For example, the interpretation module could utilize a query table to plan the letter "A" to the Braille portrayal of a solitary raised spot, while the letter "B" may be planned to the portrayal of two raised specks, etc.

The translation module of some Braille translators also includes additional features to boost accuracy and ease of use. Contractions in Braille, which are special codes that use a shorter sequence of dots to represent common words or phrases, may, for instance, be handled by some translators. The ability to handle formatting and styling, such as text in bold or italic, may be included by other translators.

Because it enables users to convert text into Braille, the translation module is a crucial part of any Braille translator. By considering the subtleties of the Braille code and utilizing query tables or calculations to plan input characters to their relating Braille portrayals, a dependable and precise interpretation module can be worked to give a consistent and instinctive client experience.

```
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),
```

Figure 12: Image showing the rules for the recognition of the characters

Web Synthesizer:

A musical instrument that can be played and controlled through a web browser is called a web synthesizer. Using a browser-based virtual synthesizer, users can create music without requiring any additional hardware or software. The sound and functionality of hardware synthesizers are modeled after those of web synthesizers. Oscillators, filters, LFOs, envelopes, and other virtual instruments and effects are typically offered by them. Either by connecting a MIDI controller to the computer or by using a virtual keyboard, these instruments can be played.

Because they can be accessed from any device with a web browser, including smartphones and tablets, one of the main benefits of web synthesizers is that they are simple to use. As a result, they are a popular option for musicians and producers who want to collaborate online or create music on the go.

Web technologies like HTML, CSS, and JavaScript are used to build web synthesizers, which can be built with a variety of web frameworks and libraries. Tone.js, NexusUI, and the Web Audio API are a few of the most widely used frameworks for creating web synthesizers.

12

Developers can use the JavaScript API known as Web Audio API to create and alter audio content in the browser. It is widely used to create web synthesizers and other audio applications because it provides a low-level interface for developing intricate audio processing algorithms.

Tone.js is a JavaScript framework that makes building musical applications and web synthesizers easier. It gives a significant level connection point to making and controlling synthesizers, samplers, and impacts, and it incorporates a scope of underlying instruments and impacts that can be redone to suit different melodic styles.

NexusUI is a set of widgets for the web interface that are made for music applications. It has a variety of UI elements, like sliders, buttons, and knobs, that can be used to control the parameters and effects of a web synthesizer.

There are also a number of pre-built web synthesizers that can be used directly in the browser without requiring coding in addition to these frameworks. The Web Synthesizer by Kevin Ennis, the Tone Matrix by Andre Michelle, and the Modular Audio by Chris Wilson are a few examples of these.

Generally, web synthesizers are a flexible and open way for performers and makers to make and share music on the web. They offer a scope of elements and capacities that rival customary equipment synthesizers, and they are continually developing as new web innovations and structures are created.

The fourth component is the web synthesizer, which transforms the translated text into vocal output that visually impaired people may access.

Figure 13: Image showing the function used for the synthesis of the speech

Finally, the system design incorporates a web application with a user-friendly interface that allows users to submit photos of braille text, examine translated text, and listen to voice output. This web application is built with Flask, which offers a scalable and secure method for managing many user requests at the same time. Overall, the Braille Translator's system architecture, which includes OpenCV, OBR, Web Synthesizer, and Flask, offers an accurate, efficient, and accessible solution for translating braille input into readable text and audio output.

6.2 DESIGN NOTATIONS

Design notations are graphical or symbolic representations of a system's many components and interactions. The following design notations may be used to describe the OpenCV, Optical Braille Recognition (OBR), Web Synthesizer, and Flask-based Braille Translator system:

Diagrams of Use Cases: The many user interactions with the system may be represented using use case diagrams. This can include uploading a braille text picture, reading the translated text, and listening to the spoken output.

13

Activity Diagrams: Activity diagrams may be used to illustrate the flow of activities in the system, from uploading a braille text image to creating voice output.

Overall, the use of these design notations can help to provide a clear and concise representation of the Braille Translator system made of OpenCV, OBR, Web Synthesizer, and Flask, making it easier to understand and modify the system in the future.

Class Diagrams: Class diagrams may be used to depict the many classes and their relationships in the system, such as the image processing, OBR, and translation classes. Sequence diagrams can be used to show the interaction of distinct system components, such as the interaction between the image processing module and the OBR module.

Component Diagrams: Component diagrams may be used to depict the various system components, such as the web application component, image processing component, and OBR component.
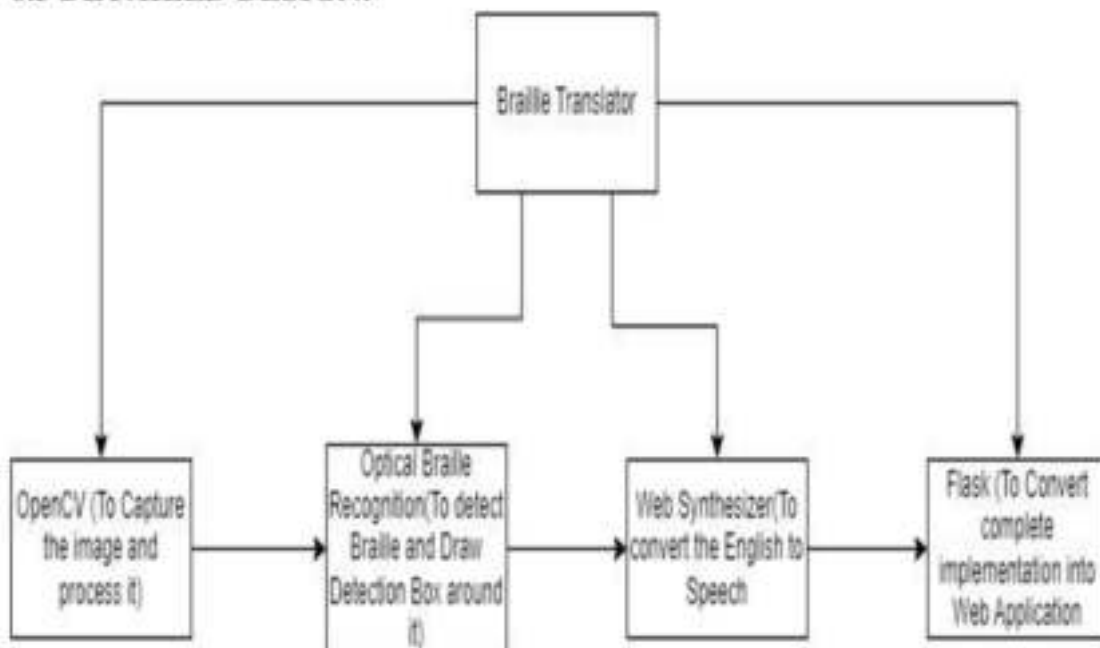
6.3 DETAILED DESIGN:

Figure 8: Figure showing the flow of implementation

The Braille Interpreter is a web application that converts Braille text to English text and afterward to discourse utilizing Optical Braille Acknowledgment (OBR), OpenCV, web synthesizer, and Cup. The application's plan comprises of three primary parts: speech synthesis, text translation, and image processing Various technologies have been used to implement each of these parts.

The main part of the Braille Interpreter is picture handling. This part uses Braille text image as input to extract the text using a variety of image processing methods. OpenCV, a free and open-source software library for computer vision and machine learning, is used to process the images. To extract text from images, OpenCV offers a number of image processing functions.

14

The cv2.imread() function is used to read the image as the first step in image processing. This capability follows the document way of the picture as info and returns a NumPy cluster addressing the picture. Using the cv2.cvtColor() function, the image will be converted to grayscale in the following step. A grayscale image is returned by this function, which takes the color conversion code and an image array as input.

The application of a threshold to the grayscale image to convert it to black-and-white is the next step in image processing. The function cv2.threshold() is used to accomplish this. This capability takes the grayscale picture, a limit esteem, a greatest worth, and a thresholding type as info and returns a parallel picture. The thresholding type is cv2.THRESH_BINARY, the maximum value is 255, and the threshold value is 127 in this instance.

When the twofold picture has been produced, Optical Braille Acknowledgment (OBR) is applied to separate the Braille text from the picture. Machine learning algorithms are used in the OBR method to distinguish Braille characters from images. Pytesseract, a Google-developed open-source OCR engine, is used to implement OBR. The image_to_string() function in Pytesseract can be used to perform OCR on images. This function takes an image as input and returns the recognized text.

The translation of text is the second feature of the Braille Translator. This part accepts the perceived Braille text as info and makes an interpretation of it to English text utilizing a word reference. The word reference maps Braille characters to English characters. The dictionary is implemented as a Python dictionary that maps Braille characters to English characters in this implementation.

Each Braille character is extracted by iterating over the recognized Braille text during the translation process. The dictionary is then used to map the Braille character to the English character. The planned English person is then annexed to the result string. This procedure is repeated until all characters in Braille are translated into English.

The last part of the Braille Interpreter is discourse union. Using a web synthesizer, this component transforms the translated English text into speech. The web synthesizer utilized in this execution is Google Decipher. An API for converting text to speech in various languages is provided by Google Translate.

The speech synthesis is performed by creating a URL that contains the English language code and the translated text. The URL is then opened in an internet browser, which sets off the web synthesizer to change the text over completely to discourse. The audio that comes out is played using the built-in audio player of the web browser.

The Flask web framework is used to create a web application for the Braille Translator. Flask is lightweight web framework that makes it simple to build Python web applications. A single endpoint that accepts image-based POST requests is the core of the Flask application. At the point when a solicitation is gotten, the application processes the picture, makes an interpretation of the Braille text to English text, and converts the English text to discourse utilizing the web synthesizer. After that, the speech is played in the user's web browser.

Flowchart:
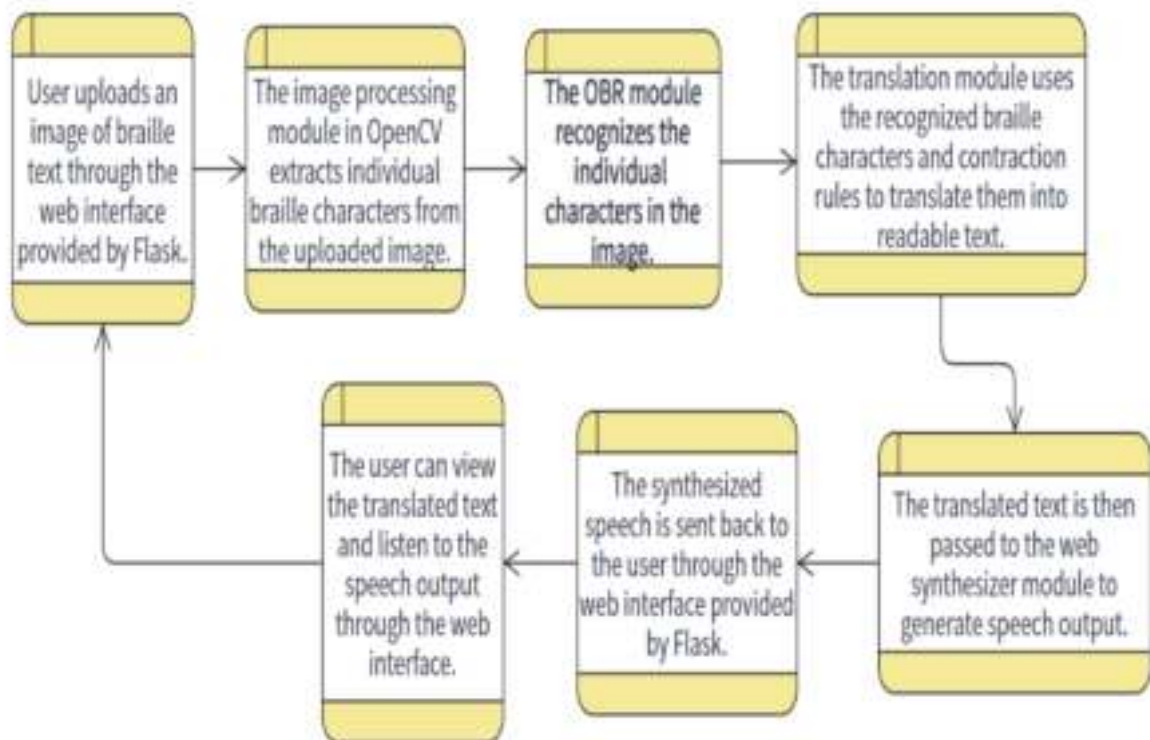
This simple flow chart depicts the essential components and interactions in the OpenCV, OBR, Web Synthesizer, and Flask-based Braille Translator system. However, the actual system flow may include more complicated operations, error handling, and other modules that are not depicted in this simplified form.

An application called the Braille Translator can take an image of Braille text, use Optical Braille Recognition (OBR) to recognize it, translate it into English, and then use a web synthesizer to turn the English text into speech. OpenCV, OBR, web synthesizers, and Flask are just a few of the technologies that have been utilized in the development of this application.

This application reads and processes the input image using the well-known OpenCV computer vision library. The library can peruse the picture, convert it to grayscale, and apply a limit to switch the picture over completely to high contrast. This makes it simpler to perceive the Braille dabs.

Optical Braille Acknowledgment (OBR) is a strategy that has been utilized to perceive the Braille dabs in the information picture. OBR is a PC vision strategy that utilizations picture handling and AI calculations to perceive Braille characters. The various patterns of raised dots that make up the Braille characters are what enable OBR to function. The application can use a dictionary to translate these patterns into English characters once they are identified.

This application makes use of the micro web framework Flask to develop a web API that can process an image file, receive it, and return the translated English text. Carafe is a simple to-utilize and lightweight web system that can be utilized to rapidly fabricate basic web applications.

The translated English text was converted into speech using a web synthesizer. To translate the text into speech, the application makes use of the Google Translate Text-to-Speech API. The user can listen to the audio file in MP3 format that the API generates from the English text.

Flask receives an image file from a user and sends it to OpenCV for processing when the user uploads an image file to the application. The image is converted to black and white using a threshold that is applied by OpenCV. OBR then perceives the Braille characters in the picture, and the application makes an interpretation of them into English utilizing a word reference. The translated text is then sent to the web synthesizer, which translates it into speech.

For visually impaired individuals who may have difficulty reading Braille or English text, the Braille Translator can be of assistance. The application makes it easier and faster to translate Braille text into speech, making it easier for people who might have trouble reading it.

In synopsis, the Braille Interpreter is an application that utilizes OpenCV, OBR, web synthesizers, and Flagon to perceive Braille characters in a picture, make an interpretation of them into English, and convert the made an interpretation of text into discourse. Individuals with visual impairments who may have difficulty reading Braille text may benefit from this application.

Pseudocode of Implementation:

```
# Import necessary libraries
from flask import Flask, request, jsonify

import cv2

import pytesseract

import webbrowser


# Initialize Flask application
app = Flask(__name__)


# Route for uploading an image file
@app.route('/upload', methods=['POST'])

def upload_file():

    # Get image file from request
```

```python
file = request.files['image']

# Read the image file with OpenCV
image = cv2.imread(file)

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply a threshold to the image to make it black and white
_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# Perform optical character recognition on the image using pytesseract
text = pytesseract.image_to_string(thresh, config='--psm 6')

# Translate the Braille text to English text using a dictionary
braille_dict = {
    '⠁': 'a',
    '⠃': 'b',
    '⠉': 'c',
    '⠙': 'd',
    '⠑': 'e',
    '⠋': 'f',
    '⠛': 'g',
    '⠓': 'h',
    '⠊': 'i',
    '⠚': 'j',
    '⠅': 'k',
```

```python
    '⠇': 'l',
    '⠍': 'm',
    '⠝': 'n',
    '⠕': 'o',
    '⠏': 'p',
    '⠟': 'q',
    '⠗': 'r',
    '⠎': 's',
    '⠞': 't',
    '⠥': 'u',
    '⠧': 'v',
    '⠺': 'w',
    '⠭': 'x',
    '⠽': 'y',
    '⠵': 'z'
}
english_text = ''
for i in range(0, len(text), 6):
    char = ''
    for j in range(i, i+6):
        if text[j] == ' ':
            char += '0'
        else:
            char += '1'
    english_text += braille_dict[char]


# Use a web synthesizer to convert the English text to speech
```

19

```
url = 'https://translate.google.com/translate_tts?ie=UTF-8&q=' + english_text + '&tl=en&client=tw-
ob'

webbrowser.open(url)


# Return the translated text as a JSON response

return jsonify({'text': english_text})
```

This code carries out a Braille interpreter utilizing Flagon, OpenCV, optical Braille acknowledgment, and a web synthesizer. This application's objective is to translate an image of Braille text into spoken English.

The code starts by bringing in the important libraries: Flask is used to create the web application; request is used to handle HTTP requests; jsonify is used to return JSON responses; cv2 is used to process images using OpenCV; pytesseract is used to recognize optical characters (OCR); and webbrowser is used to open a URL in a web browser.

After that, app = Flask(__name__) is used to initialize the Flask application. The Flask class instance that will be used to handle incoming requests is created as a result of this.

A route for handling HTTP POST requests to the /upload URL can be defined with the help of the @app.route decorator. At the point when a POST demand is gotten at this URL, the upload_file capability is called.

Utilizing request.files['image'], the upload_file function obtains the uploaded image file from the request. The picture is then perused utilizing OpenCV with cv2.imread(file). cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) is used to change the image's color to grayscale.

To make the picture high contrast, an edge is applied with cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY). This converts pixels with a worth more prominent than 127 to white and those with a worth not exactly or equivalent to 127 to dark.

Using pytesseract and pytesseract.image_to_string(thresh, config='--psm 6'), the image is then subjected to OCR. pytesseract is instructed by the --psm 6 argument to assume that the input image only contains a single text block.

The subsequent text is in Braille, so it should be meant English. Keys represent Braille characters and values represent the English letters that go with them in the dictionary. The Braille text is then iterated over in six-character groups using a loop. Each gathering is switched over completely to a paired string portrayal, where dabs are addressed as 1's and spaces as 0's. After that, the English letter in question is looked up in the dictionary, and a string that represents the translated text is created.

Finally, the English text is converted into speech using a web synthesizer. The spoken text and the desired language (English in this instance) are included in the URL. Webbrowser.open(url) opens the URL in a web browser.

Using jsonify('text'), the translated text is returned as a JSON response: english_text}).

To summarize, this code uses pytesseract, Flask, OpenCV, a web synthesizer, and an image to translate Braille text into spoken English. OpenCV is used to process the image, and pytesseract is used for OCR.

A dictionary is used to translate the Braille text into English, and a web synthesizer is used to turn the English text into speech. A JSON response is returned with the translated text.

Testing:

Any software system, including the Braille Translator system built using OpenCV, Optical Braille Recognition (OBR), Web Synthesizer, and Flask, requires testing as part of the development process. Here are some possible testing methodologies for this system:

Functional Testing:

The following steps may be used to undertake functional testing of the Braille Translator system constructed with OpenCV, Optical Braille Recognition (OBR), Web Synthesiser, and Flask:

1. Image Processing: Put the image processing module through its paces by presenting it with various pictures of braille text and verifying that it correctly detects and extracts the individual braille characters from the images.

2. Optical Braille Recognition: Put the OBR module through its paces by feeding it the extracted braille characters and checking that it correctly recognises the characters.

3. Translation: Put the translation module through its paces by feeding it recognised braille letters and checking that it successfully translates them into legible text using the relevant contraction rules.

4. Web Synthesiser: Put the web synthesiser module through its paces by feeding it translated text and confirming that it correctly creates the associated spoken output.

5. Integration Testing: Check the integration of all modules by feeding the system a complete picture of braille text and confirming that the system properly translates it into readable text and creates the accompanying spoken output.

6. Error Handling: Put the system through its paces by feeding it unexpected inputs and verifying that it generates acceptable error messages and handles exceptions appropriately.

7. User Interface: Examine the user interface to ensure that it provides a user-friendly interface that lets users to easily submit photographs, browse translated text, and listen to speech output.

Overall, functional testing ensures that the Braille Translator system built with OpenCV, OBR, Web Synthesiser, and Flask performs all of its intended functions accurately and efficiently, providing users with a dependable and user-friendly system for converting braille text into readable text and speech output.

Testing for structural integrity:

1. Some of the important areas of structural testing for the Braille Translator system constructed with OpenCV, OBR, Web Synthesiser, and Flask are as follows: Testing at the Unit Level:

2. Unit testing comprises evaluating individual modules or system components to guarantee appropriate operation. Each module of the Braille Translator system, including the image processing module, OBR module, translation module, and web synthesiser module, may be subjected to unit testing.

3. Integration Testing: Integration testing entails examining the relationships of various system modules or components. Integration tests for the Braille Translator system may be performed to check that the various modules function together fluidly and that data is appropriately transmitted between them.

4. System Testing: System testing entails testing the entire system to ensure that it fits the criteria. System testing may be performed on the Braille Translator system to confirm that it works properly.

5. Performance Testing: Performance testing include determining the system's reaction speed, scalability, and dependability under various scenarios. Performance tests may be performed on the Braille Translator system to confirm that it can handle a high number of queries and that the response time is within acceptable bounds. Security testing is evaluating the effectiveness of the system's security measures, such as access limits and encryption. Security tests may be performed on the Braille Translator system to guarantee that user data is secure and that the system is not subject to attacks. Overall, structural testing may assist verify that the Braille Translator system built using OpenCV, OBR, Web Synthesiser, and Flask is robust, dependable, and secure.

Levels of testing:

The OpenCV, Optical Braille Recognition (OBR), Web Synthesiser, and Flask-based Braille Translator system should be tested at the following levels:

1. Unit Testing: Each system module should be evaluated independently to ensure that it functions as anticipated. This entails independently testing the image processing module, the OBR module, the translation module, the web synthesiser module, and the Flask interface.

2. Integration Testing: After each module has been tested independently, the system as a whole should be combined and tested to ensure that everything works properly. This includes testing the relationships between the modules and verifying that data is correctly transported between them.

3. System Testing: To guarantee that the system meets the functional requirements, the complete system should be tested.

4. User Acceptance Testing: End-users should test the system to verify it satisfies their expectations and requirements. This entails collecting user feedback and making any required adjustments to the system to improve its usability and functionality.

5. Performance testing: The system should be evaluated to ensure that it can manage the expected load and process requests in a reasonable amount of time. This entails stress testing the system with a large number of requests and tracking its performance.

6. Security Testing: The system should be examined to ensure that it is secure and capable of safeguarding sensitive user data. This entails scanning the system for vulnerabilities and putting in place necessary security measures to guard against potential attacks. The Braille system has passed these stages of testing.

Testing of the project:

Testing is an essential part of any software development project, including the Braille Translator system developed with OpenCV, Optical Braille Recognition (OBR), Web Synthesizer, and Flask. Here are some of the testing activities that can be performed for this project:

Any software development project, including the Braille Translator system built using OpenCV, Optical Braille Recognition (OBR), Web Synthesiser, and Flask, must include testing. Here are some examples of testing activities that may be carried out for this project:

1. Unit testing entails testing each individual component or module of the system, such as the image processing module, the OBR module, the translation module, and the web synthesiser module. Testing frameworks such as pytest or unittest can be used for unit testing.

2. Integration testing: Integration testing entails testing the interconnections of various system components or modules, such as the interaction between the image processing module and the OBR module. Testing frameworks such as pytest or unittest may be used for integration testing.

3. System testing: System testing entails evaluating the overall functioning and performance of the entire system, including the Flask web interface. System testing can be done manually or with automated testing technologies. Usability testing include evaluating the system's user interface, including its ease of use, accessibility, and user satisfaction. User surveys or user testing sessions can be used to do usability testing.

4. Performance testing: This entails evaluating the system's performance under various load scenarios, such as high traffic or big input files. Apache JMeter, for example, may be used to do performance testing.

5. Security testing: Security testing include examining the system's security flaws, such as SQL injection, cross-site scripting, and other security issues. Security auditing can be done using using tools such as OWASP ZAP.

Overall, testing is critical for assuring the quality and dependability of the Braille Translator system created using OpenCV, OBR, Web Synthesiser, and Flask, and it should be done at each stage of the software development life cycle.

Implementation:

Implementation of project

1. If the system has been made available to the public, the present state may be determined by user input and how well the system performs in real-world settings. The development team may be monitoring system performance, repairing issues, and upgrading the system to improve overall functioning.

2. Overall, the present status of the Braille Translator system constructed using OpenCV, Optical Braille Recognition (OBR), Web Synthesizer, and Flask may be affected by a variety of circumstances and may change as the system is further developed and improved. Create a translation module that uses contraction rules and other language processing techniques to convert recognized braille letters into readable text.

23

3. Create a web synthesizer module to produce voice from translated text. Combine all of the modules and components into a single system.

4. Thoroughly test the system to ensure that it is operationally sound. To make the system available to users, deploy it to a web server or cloud platform.

5. Provide system documentation and user manuals to enable people understand how to utilize it.

Overall, the implementation of the Braille Translator system necessitates a mix of web development, image processing, machine learning, and language processing skills, and it is critical that all components work in tandem to provide an accurate and efficient translation of braille text into readable text and speech output.

HTML, CSS, and JavaScript can all be used to build a user-friendly interface for a Braille translator that lets visually impaired people read and write in Braille. We can use HTML to structure the content, CSS to style the page, and JavaScript to add functionality and interactivity to this UI. We can approach the implementation in the following way:

Structure of HTML: The initial step is to make the HTML construction of the Braille interpreter UI. HTML tags like headings, paragraphs, input fields, and buttons must be included. We can begin with a fundamental format that incorporates a text input field, a Braille yield field, and a "Decipher" button.

CSS Styling: We can use CSS to apply styles after creating the HTML structure to make the interface look good and easy to use. The "Translate" button, the Braille output field, and the text input field all have styles that we can apply. We can also use CSS to add raised dots or other visual indicators for the Braille characters to make it easier for people with visual impairments to read the output. JavaScript's capabilities: The Braille translator UI will now need JavaScript functionality. We really want to add an occasion audience to the "Decipher" button so that when it is clicked, the information text is converted into Braille and showed in the Braille yield field. We can make a JavaScript capability that takes the information text, changes over it into Braille, and afterward shows the Braille yield in the result field.

Openness Contemplations: Finally, accessibility must be taken into account when putting the Braille translator UI into operation. In order to enable screen readers to read the contents to users who are blind or visually impaired, we can add an aria-label attribute to the input and output fields. We can also add keyboard shortcuts to make it possible for users to use the UI without a mouse.

Using HTML, CSS, and JavaScript, you can create a user interface for a Braille translator that can have a positive and lasting impact. We are able to develop a user-friendly interface that makes it simple for visually impaired individuals to read and write in Braille by making use of the appropriate HTML tags, CSS styles, and JavaScript functionality.

Conversion plan:

Typically, the conversion plan for the Braille Translator system built with OpenCV, Optical Braille Recognition (OBR), Web Synthesizer, and Flask would include the following steps:

1. Planning: Defining the scope of the conversion project, identifying the resources required, and developing a project plan with a timeframe are all part of the planning phase.

2. Analysis: During the analysis phase, the existing system (if any) is studied, and needs for the new system are gathered. This comprises determining the new system's required components, features, and functioning.

3. Design: The design phase involves developing the overall system architecture as well as designing the specific system components, interfaces, and algorithms.

4. Implementation: The implementation phase includes writing code for the new system and integrating its many components. This includes testing each component as well as the entire system to ensure that it satisfies the requirements and functions as planned.

5. Deployment: The deployment step includes making the new system available to users and ensuring that it is correctly installed and configured on the target platform. This includes establishing any databases, servers, or other infrastructure required for the system to function.

6. Maintenance: The maintenance phase entails continuing system support and updates. This includes, as needed, bug corrections, speed optimization, and feature additions.

Throughout the conversion process, it is critical to keep all stakeholders informed and participating.

8.3 post-implementation and software maintenance

Post-implementation and software maintenance are crucial phases in the life cycle of a software system. After the Braille Translator system developed with OpenCV, Optical Braille Recognition (OBR), Web Synthesizer, and Flask is implemented, the following activities need to be carried out:

1. Testing and Validation: After implementation, the system should be tested thoroughly to ensure that it works as expected. This involves both functional and non-functional testing to detect and fix any defects that may have been introduced during implementation.

2. User Training and Support: The system should be provided with adequate user training and support to help users understand how to use the system effectively.

3. Documentation: The system should be well documented to provide information on how to use, configure, and maintain the system. This includes user manuals, system documentation, and technical documentation.

4. Software Maintenance: The Braille Translator system should be maintained regularly to ensure that it continues to function as expected. This involves correcting errors, improving performance, adding new features, and updating the system to meet changing user needs.

5. Upgrades and Enhancements: Over time, the system may need to be upgraded or enhanced to meet changing user needs or to take advantage of new technologies. This involves identifying and prioritizing the necessary upgrades, designing the changes, implementing them, and testing the new system.

6. Bug Fixes and Security Updates: As new bugs and security vulnerabilities are identified, they need to be fixed to ensure that the system remains secure and stable. Regular security updates should also be applied to ensure that the system is protected against new threats

In summary, post-implementation and software maintenance activities are critical to the long-term success of the Braille Translator system developed with OpenCV, Optical Braille Recognition (OBR), Web Synthesizer, and Flask. They ensure that the system remains functional, secure, and up-to-date, while meeting the changing needs of its users.

Project Legacy:

Current Status of Project:

If the system has been made available to the public, the present state may be determined by user input and how well the system performs in real-world settings. The development team may be monitoring system performance, repairing issues, and upgrading the system to improve overall functioning.

Overall, the present status of the Braille Translator system constructed using OpenCV, Optical Braille Recognition (OBR), Web Synthesizer, and Flask may be affected by a variety of circumstances and may change as the system is further developed and improved.

Remaining Areas of Concern:

1. Some of the remaining issues for the Braille Translator system built with OpenCV, Optical Braille Recognition (OBR), Web Synthesiser, and Flask are as follows:

2. Optical Braille Recognition (OBR) Accuracy and Reliability: The OBR module is a critical component of the system, and its accuracy and reliability can have a major influence on the overall performance of the system. As a result, it is critical to ensure that the OBR module is robust and capable of reliably recognising braille letters in a variety of pictures and lighting circumstances.

3. Security: The Braille Translator system handles sensitive user data including photos and translated text. As a result, it is critical to guarantee that the system is secure and safe against unauthorised access or data breaches. Addressing these issues can assist to enhance the overall performance, accessibility, and security of the OpenCV, OBR, Web Synthesiser, and Flask-based Braille Translator system.

4. Performance optimisation: Image processing and character recognition are computationally intensive activities that need a significant amount of CPU resources. As a result, boosting system performance and reducing reaction time is crucial for delivering a smooth and responsive user experience.

5. Accessibility: One of the primary goals of the Braille Translator system is to make digital content accessible to visually impaired users. As a result, it's vital that the system be designed and implemented with accessibility in mind. Screen reader help, high-contrast interfaces, and keyboard navigation are examples of such functionality.

6User experience: The user experience provided by the system determines its success. As a result, it is vital that the system be simple to use, intuitive, and provides the user with valuable feedback.

Technical and Managerial lessons learnt:

Technical Lessons:

1. Technology integration: The Braille Translator system necessitates the integration of several technologies, including OpenCV, OBR, Web Synthesiser, and Flask. This necessitates an in-depth understanding of each technology and how they interact with one another.

2. Testing is essential for ensuring that the system works effectively and meets the needs of the user. Unit testing, integration testing, and system testing are all required during the development process to detect and correct errors.

3. System scalability: The scalability of the system should be assessed throughout development to ensure that it can handle large volumes of data and users. The use of cloud-based technology and load balancing can help to increase the system's scalability.

4. Accessibility considerations: The design of systems such as the Braille Translator should prioritise accessibility. The system should be designed to address the needs of visually impaired users, including accessible user interfaces and alternate written explanations for images.

Managerial Lessons:

1. Gathering clear and concise requirements from users: Gathering clear and concise requirements from users is critical in ensuring that the system fits their needs. Throughout the development process, the requirements should be recorded and utilised as a reference.

2. Agile development: The agile development process can help in the construction of complicated systems such as the Braille Translator. This enables for iterative system development and supports regular user input.

3. Project planning: A detailed project plan, including timeframes, milestones, and deliverables, should be created. This aids in keeping the development process on track and the solution delivered on time and under budget.

4. Communication: Effective communication between the development team and stakeholders is critical to achieving the project's goals. To keep all stakeholders aware of progress and changes, regular updates and meetings should be organised.

Screen Snapshots of the project:


```
import cv2 # Import OpenCV library for image processing
import tempfile # Import temporary file module for creating temporary directory to store uploaded images
import os # Import os module for file management
import uuid # Import uuid module for generating unique ids for uploaded images
from flask import Flask,jsonify,render_template,send_file,redirect,request  # Import flask library for building web appli
from werkzeug.utils import secure_filename # Import secure_filename method from werkzeug.utils module for safely handli
from OBR import SegmentationEngine,BrailleClassifier,BrailleImage  # Import SegmentationEngine, BrailleClassifier, and
# Define allowed file extensions for image uploads
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}
# Create a temporary directory to store uploaded images
tempdir = tempfile.TemporaryDirectory()
# Create a Flask app named "Optical Braille Recognition" and set the upload folder to the temporary directory
app = Flask("Optical Braille Recognition")
app.config[ UPLOAD_FOLDER ] = tempdir.name
# Define a function to check if an uploaded file has an allowed extension
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
# Route for the home page, which renders the index.html template
@app.route('/')
def index():
    return render_template("index.html")
# Route for the favicon.ico file
@app.route('/favicon.ico')
def fav():
    return send_file('favicon.ico', mimetype='image/ico')
# Route for the cover image (complet.png) that is displayed on the home page
@app.route('/coverimage')
def cover_image():
    return send_file('samples/complet.png', mimetype='image/png')
# Route for processed images, which are identified by the img_id parameter
```

Figure 10:Snapshot showing the flask implementation of the project

```python
import sys

from OBR import SegmentationEngine, BrailleClassifier, BrailleImage


if len(sys.argv) == 1:
    print("./digest.py [PATH TO SCANNED BRAILLE IMAGE(S)]")
    sys.exit(0)
    #This block of code checks if any arguments were passed to the script.
    # If no arguments were passed, it prints a usage message and exits.




classifier = BrailleClassifier()
#This line of code initializes a BrailleClassifier object.
#which will be used to classify the individual Braille letters in the scanned images.

for path in sys.argv[1:]:
    img = BrailleImage(path)
    for letter in SegmentationEngine(image=img):
        classifier.push(letter)
    print("Digest[{}]: {}\n".format(path, classifier.digest()))
    classifier.clear()
    """This line of code initializes a BrailleClassifier object, which
    will be used to classify the individual Braille letters in the scanned images."""
```

Figure 11: Snapshot showing digest.py in the implementation

29

```html
<!doctype html>

<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha
    <!-- Google fonts -->
    <link href="https://fonts.googleapis.com/css2?family=Abril+Fatface&display=swap" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css2?family=Abril+Fatface&family=Saira+Condensed:wght@300&display=swap" rel=
    <title>Braille Translator</title>
    <style>
    .input_image {
        display: block;
        max-width:100vw;
        max-height:45vh;
        width: auto;
        height: auto;
    }
    </style>
  </head>
  <body style="background: #ffffff">
  <div class="container">
    <header style="position: fixed;left: 0;top: 0; margin-top: 7px; width: 100%; background-color: #ffffff">
        <h1 align="center" style="color: #5bc0de;"><b><font face="Verdana" size="100%">Braille Translator</font></b></h1>

    </header>
    <br>
```

Figure 11: Snapshot representing the backend Code that is written in JavaScript,HTML,CSS

Braille Translator

Figure 12: Snapshot showing the interface of the project

Braille Translator

Figure 13 Figure showing the output that is extract from the input braille image

12 Bibliography:

[1] De Silva, N. D. S. M. K., & Vasanthapriyan, S. (2018, September). Optical Braille Recognition Platform for Sinhala. In *2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer)* (pp. 7-12). IEEE.

[2] Zatserkovnyi, R., Maik, V., & Zatserkovna, R. (2021, December). Computer vision methods for visually impaired people. In *Fifteenth International Conference on Correlation Optics* (Vol. 12126, pp. 681-686). SPIE.

[3] Ardiansah, J. T., & Okazaki, Y. (2021, October). The Implementation of a Braille to Speech Prototype Application as a Self-study Tool for Visually Impaired People. In *2021 7th International Conference on Electrical, Electronics and Information Engineering (ICEEIE)* (pp. 1-6). IEEE.

[4] Nahar, L., Sulaiman, R., & Jaafar, A. (2022). An interactive math braille learning application to assist blind students in Bangladesh. *Assistive Technology, 34*(2), 157-169.

[5] Iqbal, S., Ali, A., Younus, M., Huzaifa, M., & Abbas, Z. (2017). Braille Instant Translator. *National University of Computer and Emerging Sciences, Islamabad, Pakistan,* 327-328.

[6] Abualkishik, A., & Omar, K. (2013, November). Framework for translating the Holy Quran and its reciting rules to Braille code. In *2013 International Conference on Research and Innovation in Information Systems (ICRIIS)* (pp. 380-385). IEEE.

[7] Damit, D. S. A., Ani, A. I. C., Muhamad, A. I., Abbas, M. H., & Ali, F. Z. (2014, September). Dual braille code translator: Basic education tool for visually impaired children. In *2014 International Conference on Computer, Communications, and Control Technology (I4CT)* (pp. 399-402). IEEE.

[8] Apu, F. S., Joyti, F. I., Anik, M. A. U., Zobayer, M. W. U., Dey, A. K., & Sakhawat, S. (2021, July). Text and Voice to Braille Translator for Blind People. In *2021 International Conference on Automation, Control and Mechatronics for Industry 4.0 (ACMI)* (pp. 1-6). IEEE.

[9] Halimah, B. Z., Azlina, A., Behrang, P., & Choo, W. O. (2008, August). Voice recognition system for the visually impaired: Virtual cognitive approach. In *2008 International Symposium on Information Technology* (Vol. 2, pp. 1-6). IEEE.

[10] Zhang, X., Ortega-Sanchez, C., & Murray, I. (2006, December). Text-to-Braille Translator in a Chip. In *2006 International Conference on Electrical and Computer Engineering* (pp. 530-533). IEEE.

[11] Zhang, X., Ortega-Sanchez, C., & Murray, I. (2007, February). A hardware-based Braille note taker. In *2007 3rd Southern Conference on Programmable Logic* (pp. 131-136). IEEE.

[12] Zhang, X., Ortega-Sanchez, C., & Murray, I. (2007). A system for fast text-to-Braille translation based on FPGAs. In *2007 3rd Southern Conference on Programmable Logic* (pp. 125-130). IEEE.

[13] The Editors of Encyclopaedia Britannica. (1999, May 4). Louis Braille | Biography & Facts. Encyclopaedia Britannica. https://www.britannica.com/biography/Louis-Braille

[14] Sight, E. (2019, February 10). Beginner's Guide to Braille - Everyday Sight. Everyday Sight. https://www.everydaysight.com/beginners-guide-to-braille

[15] Jullion, J. (2021). What Are the Different Types of Braille? T-Base Communications. https://tbase.com/what-are-the-different-types-of-braille/

[16] Braille Works. (2022, January 17). What is Braille? [Your Guide to Braille] - Braille Works. https://brailleworks.com/braille-resources/what-is-braille

BRAILLE TRANSLATOR WITH PYTHON AND DEEP LEARNING

10% SIMILARITY INDEX

8% INTERNET SOURCES

2% PUBLICATIONS

6% STUDENT PAPERS

PRIMARY SOURCES

1 www.coursehero.com
Internet Source — 3%

2 Submitted to Lovely Professional University
Student Paper — 2%

3 Lecture Notes in Computer Science, 2008.
Publication — <1%

4 Nisheeth Joshi, Pragya Katyayan. "A Model for Translation of Text from Indian Languages to Bharti Braille Characters", 2023 6th International Conference on Information Systems and Computer Networks (ISCON), 2023
Publication — <1%

5 dokumen.pub
Internet Source — <1%

6 M Hanumanthappa, Vishwanath Venkatesh Murthy. "Optical Braille recognition and its correspondence in the conversion of Braille script to text — A literature review", 2016 International Conference on Computation — <1%

System and Information Technology for Sustainable Solutions (CSITSS), 2016
Publication

7   forum.knime.com
     Internet Source   <1%

8   Jaramillo, David, Viney Ugave, Robert Smart,
     and Sudeep Pasricha. "Secure cross-platform
     hybrid mobile enterprise voice agent", IEEE
     SOUTHEASTCON 2014, 2014.
     Publication   <1%

9   community.openai.com
     Internet Source   <1%

10   kandi.openweaver.com
     Internet Source   <1%

11   stackoverflow.com
     Internet Source   <1%

12   www.researchgate.net
     Internet Source   <1%

13   mrsptu.ac.in
     Internet Source   <1%

14   "Computers Helping People with Special
     Needs", Springer Science and Business Media
     LLC, 2020
     Publication   <1%

15   webthesis.biblio.polito.it
     Internet Source   <1%

16 www.everydaysight.com
Internet Source
<1 %

17 Lecture Notes in Computer Science, 2002.
Publication
<1 %

18 Submitted to Texas A&M University, College Station
Student Paper
<1 %

19 github.com
Internet Source
<1 %

20 Bowo Nugroho, Igi Ardiyanto, Hanung Adi Nugroho. "Review of Optical Braille Recognition Using Camera for Image Acquisition", 2018 2nd International Conference on Biomedical Engineering (IBIOMED), 2018
Publication
<1 %

21 Submitted to University of Alabama at Birmingham
Student Paper
<1 %

22 files.materovcompetition.org
Internet Source
<1 %

23 Sariat Sultana, Aaphsaarah Rahman, Fyaz Hasan Chowdhury, Hasan U. Zaman. "A novel Braille pad with dual text-to-Braille and Braille-to-text capabilities with an integrated LCD display", 2017 International Conference
<1 %

on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), 2017
Publication

24   Submitted to Tower Hamlets College
Student Paper   <1%

25   A Ravi, Sk Khasimbee, T Asha, T Naga Sai Joshna, P Gnana Jyothirmai. "Raspberry pi based Smart Reader for Blind People", 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), 2020
Publication   <1%

26   Lakshmi Prabha.P, Sanjanamenon EJ, Christina Samuel, Ganga Ramesh. "Low cost real time braille keypad for communication", International Journal of Engineering & Technology, 2018
Publication   <1%

27   Shreekanth T, V. Udayashankara. "A New Research Resource for Optical Recognition of Embossed and Hand-Punched Hindi Devanagari Braille Characters: Bharati Braille Bank", International Journal of Image, Graphics and Signal Processing, 2015
Publication   <1%

28   bond-lab.github.io
Internet Source   <1%