eReader

Project Report

Submitted in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

Anitta Thankachan

Anjitha M

Ashline George

Teresa Abraham

Department of Electronics

Govt. Model Engineering College

Thrikkakara, Cochin - 682 021

APJ Abdul Kalam Technological University

May 2019

Bonafide Certificate

MODEL ENGINEERING COLLEGE

THRIKKAKARA, KOCHI-21

DEPARTMENT OF ELECTRONICS

A P J ABDUL KALAM KERALA TECHNOLOGICAL UNIVERSITY

This is to certify that the Project Report entitled

eReader

Submitted by

Anitta Thankachan
Anjitha M
Ashline George
Teresa Abraham

is a bonafide account of their work done under our supervision

Project Co-ordinator	**Head of Department**	**Project Guide**
Dr. Binesh T	Dr. Laila D	Smt. Sheeba P S
Associate Professor	Professor	Assistant Professor
Dept. of Electronics	Dept. of Electronics	Dept. of Electronics

Abstract

The process to take a Braille document image and convert its content into its equivalent natural language characters, is called Braille Recognition Recognition (BCR). The project 'eReader' involves a novel idea that deals with the conversion of Braille to voice using image processing and text to speech conversion. The Braille paper is scanned, enhanced and various other image processing techniques are applied to finally obtain a text input. As the Braille paper is scanned, the image obtained will show the projected and non-projected portion of the Braille letters. This image can be further enhanced, using grayscale conversion, thresholding techniques and de-noising techniques involving morphological transformations. The obtained enhanced image is further segmented, first horizontally and then vertically, to obtain a clearer image. Then each cell (corresponding to each letter) is localised and grouped to form understandable patterns so that Braille recognition can be done. Thus the correct text corresponding to the braille script is obtained. This text can be further converted to voice. The text file is saved in mp3 format using gTTS package and playes using eSpeak.

Table of Contents

List of Figures

List of Abbreviations

BCR - Braille Character Recognition

CPU - Central Processing Unit

IDE - Integrated Development Environment

LXDE - Lightweight X11 Desktop Environment

PIXEL - Pi Improved X-Window Environment Lightweight

RGB - Red Green Blue

Acknowledgement

On the recollection of so many great favors and blessings, we offer our sincere thanks to the Almighty, the Creator and Preserver.

We express my heartfelt gratitude to **Prof. (Dr.) Vinu Thomas**, Principal, Govt. Model Engineering College, Thrikkakara for providing us with excellent library facilities. We sincerely thank **Prof. (Dr.) Laila D**, Head of Department of Electronics and Communication Engineering, for her timely advice and suggestions.

We are thankful to our project coordinator **Dr. Binesh T**, Associate Professor, Department of Electronics and Communication Engineering for the full-fledged support and guidance. We would like to express our sincere gratitude to our project guide **Smt. Sheeba P S**, Assistant Professor, Department of Electronics and Communication Engineering, who has always motivated us and helped us to do everything in a better way.

Last but not the least We are thankful to one and all of the Department of Electronics and Communication Engineering and the whole library staff for their co-operation and active involvement. We also thank our colleagues for their support.

Chapter 1

Introduction

Braille Character Recognition (BCR) is a method to locate and recognize Braille document stored in an image, such as a jpeg, jpg, tiff or a gif image, and convert the text into a coded machine form such as text file. BCR converts the pixel representation of an image into its equivalent character representation. Braille recognition has a plenty of benefits which facilitate the work in our daily life as workers in-visually impaired schools and institutes. Many schools have a very large collection of paper forms and documents that is used in teaching blind students, in order to search, read or rewrite these documents by existing hand will take a very long time, and it is only natural way of seeking to automate this process.

Based on literature review studies and remarks it can be concluded that extracting information from braille paper requires accuracy in pre-processing stage. Beside that there are still ongoing research and work need to enhance the recognition performance since image taken from braille documents are exposed to high noise ratio.

The importance in this work is to ease the interface between visually impaired people, and society and help in office automation with huge saving of time and human effort success. Research on Braille recognition has progressively become one of the hot research areas. Braille recognition not only reduces time in reading or extracting information from Braille document, furthermore, to resolves the cause of people engaged in special education issues, such as school work correcting, papers marking and etc.

Previous works point out that the need for Braille reading system to assist blind people in learning Braille, in addition to, the availability of such a system will enhance communication and collaboration possibilities with visually impaired people.

Through the time a small quantity of printed books have been available in Braille, in order ,to copy such books the only method is using the manually transcription which is both costly and time consuming. Moreover by converting such books to text form will also give advantages for handling printed documents and also can be saved in lesser volume than Braille printed one. Besides that, it can be noted that BCR

method will be practical and cost effective in order to be adopted by people who need it. Furthermore, taking the advantages of using it as an easy and efficient method can be considered as a replacement to the manual job.

Another motivation behind this is by looking to the previous works on the BCR as they recommended that pre processing and extracting features require more enhancements and modifications to overcome challenges, that's due to the fact that Braille characters have no colors to be easily recognized since they are just a raised dots.

The project 'eReader' deals with 4 main objectives. They can be listed as follows:

- Design a system that recognize Braille characters and convert it to English text based on digital image processing algorithm.
- Enhance the quality of Braille recognition by reducing errors using image preprocessing algorithms.
- Develop a robust extracting and matching algorithm that results in better recognition system.
- Convert the obtained text into speech.

Chapter 2 gives the literature survey of the project, chapter 3 describes the system overview, chapter 4 deals with the hardware overview of the project, chapter 5 deals with the software used, chapter 6 deals with the software description and chapter 7 deals with the system implementation results, chapter 8 gives the conclusion.

Chapter 2

Literature Survey

2.1 Introduction to Braille

Information in written form plays an undeniably important role in our daily lives. Recording and using information encoded in symbolic form is essential. Visually impaired people face a distinct disadvantage in this respect. In order to address their need, the most widely adopted writing convention among visually impaired people is Braille. Different attempts have been made to recognize Braille writing system for different languages.As a continuation, this study explores pre-processing and feature extraction modules of English Braille recognizer. This is because of the fact that the performance of the recognizer depends mainly on the pre-processing and feature extraction schemes.

"Access to communication in the widest sense is access to knowledge, and that is vitally important for us. We do not need pity, nor do we need to be reminded that we are vulnerable. We must be treated as equals, and communication is the way we can bring this about" Louis Braille, 1841.

According to latest statistics of the World Health Organization Who more than 285 million people are visually impaired and 7 percentage of them are totally blind. Braille method appeared nearly two centuries, and due to changes in laws and regulations it became more prevalent in daily use and it is widely used in almost all countries as a teaching tool to address the special needs of blind students.

Braille is a system of raised dots that can be read with the fingers by people who are blind or who have low vision. Teachers, parents, and others who are not visually impaired ordinarily read braille with their eyes. Braille is not a language. Rather, it is a code by which many languages—such as English, Spanish, Arabic, Chinese, and dozens of others—may be written and read. Braille is used by thousands of people all over the world in their native languages, and provides a means of literacy for all.

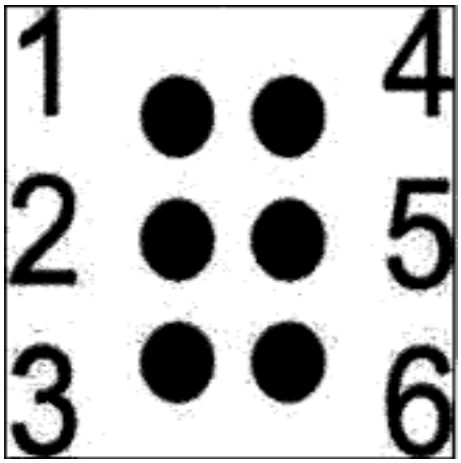


Figure 2.1: Braille cell [7]

Braille symbols are formed within units of space known as braille cells. A full braille cell consists of six raised dots arranged in two parallel rows each having three dots. The dot positions are identified by numbers from one through six. Sixty-four combinations are possible using one or more of these six dots. A single cell can be used to represent an alphabet letter, number, punctuation mark, or even a whole word. The writing on the basis of six key points three on the left and three on the right as shown in figure 2.1.

Optical Braille recognition is a computer algorithm, which attempts to automate the process of acquiring and processing Braille documents image, in order to convert it into its corresponding natural language characters, it involves two main steps: segment the Braille characters; and then convert this segmented character into its equivalent natural language character.

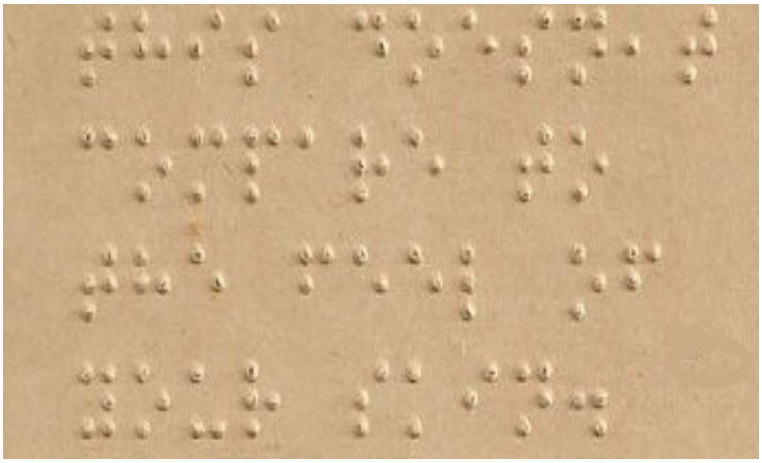


Figure 2.2: Single side braille

The dimensions of a Braille dot have been set according to the tactile resolution of the person's fingertips. The horizontal and vertical distance between dots in a character and the

distance between cells that represent a word are known. Dot height is approximately 0.02 inches (0.5 mm); the horizontal and vertical spacing between dot centers within a Braille cell is approximately 0.1 inches (2.5 mm); the blank space between dots on adjacent cells is approximately 0.15 inches (3.75 mm) horizontally and 0.2 inches (5.0 mm) vertically. A standard Braille page is 11 inches by 11.5 inches and typically has a maximum of 40 to 43 Braille cells per line and 25 lines as shown in figure 2.2 and 2.3.

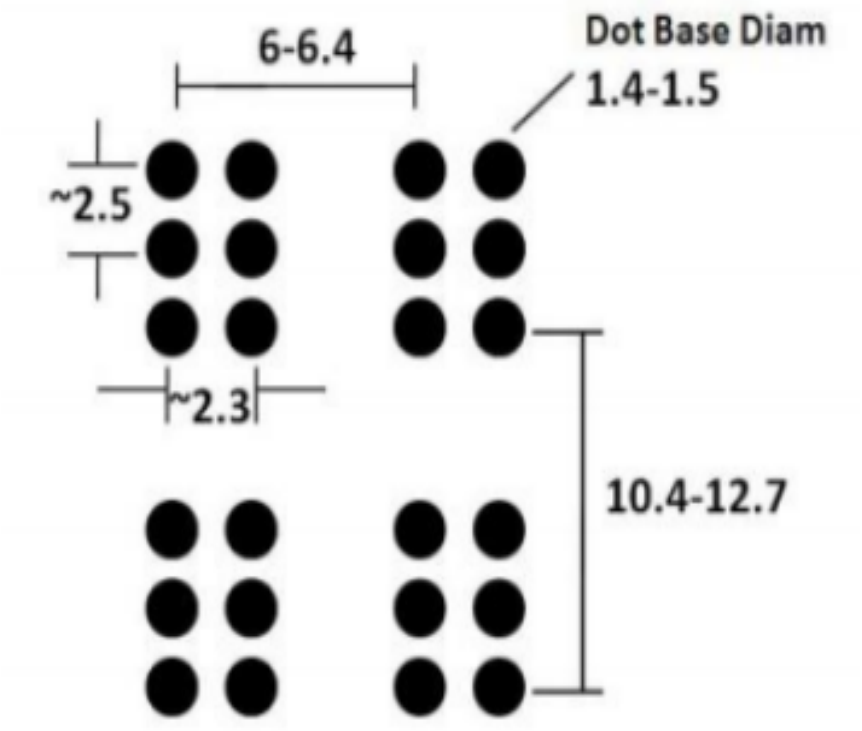


Figure 2.3: Braille cell Dimensions [7]

2.2 The Importance of Braille Recognition System

The importance of Braille recognition System to transform Braille documents to text without any knowledge of Braille reading. Thus the proposed tool is to be used in places where this conversion is necessary. For example, in school teachers need to understand the braille language in order to instruct student. Solving this issue will open up blind schools to a much wider pool of teachers.

Another approach which is more important is ability for blind student to join governmental schools and to be treated as normal student, if it becomes simple for the teacher extract information from braille document and to convert text into braille and vice versa then it will become easy to instruct blind pupils. Nowadays some governmental schools have teacher assistants' who understand braille codes and his main job is to take care of special cases students. Thus, if the ability of translating Braille into text is given to all teachers,

the issue will be solved and much wider blind student can join normal classes.

2.3 Previous Research Work

The journal [1] Braille Converter and Text-To-Speech Translator for Visually Impaired People in Sri Lanka aims to provide visually impaired people to better document controlling in their day to day life using natural language processing. It mainly consists with three main components including Text to speech, Braille converter and Language translator. Language translator, convert Sri Lankan mother languages into Braille language, such as Sinhala language into Sinhala Braille, English language into English Braille, and Tamil language into Tamil Braille. When the visually challenged people used text edition, users are capable to translate those words using mother language translator to translate the natural languages they prefer. Under the translator Sinhala, Tamil and English language could be translated. This system helps to any new user specially the visually challenged people to overcome their real communication issues and day today works efficiency. This system is tested with using three different types of users including school principal, teachers and blind students in Rathmalana Blind School in Sri Lanka. Satisfaction rates among the tested group are high.

The technologies which are used to implement the system are developed using NetBeans 8.0.2 as an Integrated Development Environment (IDE) and Java coding, Natural language processing and Yandex translator. This system is mainly based on Java. By using java, it will reduce extra cost that need to be barred for purchasing some components and tools. Because java is open source and all components are free to use. To develop this system “java” with “NetBeans” platform is more suitable.

Main functionalities include:

a) The system is ability to read the typed text to the user at the end of the typing.

b) The system is having a convertor to convert English, Sinhala and Tamil languages into the Braille language.

c) The system is having a translator to translate text into English, Tamil and Sinhala language.

d) The system should have ability to save voice effect of typed text as mp3 format.

The journal [2] A Novel Braille Pad with Dual Text-to-Braille and Braille-to-Text capabilities with an integrated LCD Display,designed system serves as both Braille writing and reading system, so visually impaired people can enhance their Braille writing and reading skills without the assistance of a Braille teacher. The designed system takes the input through Braille keyboard and produces the Braille output in Braille display; the corresponding English characters are also displayed on the LCD and also in the laptop if it is connected. It also has the capability of reading documents as text.

The journal [3] Literary Braille Language Translator to Spanish Text",presents a method for translating texts written in Braille, which was realized with a series of steps were held with the purpose of finishing the entire project. First, a preprocessing process to the braille image were performed to adapt the character recognition contained within it. Then, the separation of the information contained in each braille cell were made in order to make the comparison with a database already established. This database contains the twenty-seven alphabet letters and the twenty punctuation signs of Spanish language. It also holds a six-position characteristic vector that identifies every single letter and sign in one only way. After having made the vectors comparison, the export of the text contained in the image by using a graphical user interface were the next stage. That allowed the person to see the transcript in (.TXT) format, for further analysis of the content.

The following steps were used to get the characteristic vector:

A. Analysis of centroids

B. Cut of lines and characters recognition

C. Characteristic vector obtaining

The journal [4] Conversion of Braille to Text in English, Hindi and Tamil Languages,proposes a method to convert a scanned Braille document to text which can be read out to many through the computer. The Braille documents are preprocessed to enhance the dots and reduce the noise. The Braille cells are segmented and the dots from each cell is extracted and converted in to a number sequence. These are mapped to the appropriate alphabets of the language. The converted text is spoken out through a speech synthesizer. The paper also provides a mechanism to type the Braille characters through the number pad of the keyboard. The typed Braille character is mapped to the alphabet and spoken out. The Braille cell has a standard representation but the mapping differs for each language. In this paper mapping of English, Hindi and Tamil are considered.

The journal [5],A kind of Braille paper automatic marking system,describes a system that recognizes Braille characters from image taken by a high speed camera to Chinese character and automatically mark the Braille paper. Machine translation algorithm and image processing and recognition technology are used and improved in the system. This paper focuses on the Braille information detection, extraction and recognition in a image, including image acquisition and preprocessing, feature extraction and Braille recognition, machine translation algorithms and automatic scoring of subjective questions.

There are basically four processes followed:

1) Image acquisition and preprocessing method

2) Identification and coding of the Braille cell Each Braille character is a set of dots arranged in two columns of three on a 3 by 2 grid. The meaning of each Braille character (the Braille cell) depends on the type of Braille encoding used.

3) Machine translation

4) Automatic marking

High-speed FPGA-based image acquisition hardware devices were developed. Image processing and recognition algorithms such as the correction algorithm, optimal threshold for algorithm, de-noising algorithm etc., were provided. Further research work needed to:

1) Optimize the algorithms to improve running speed

2) System application and promotion.

The journal [6] Image Processing Techniques for Braille Writing Recognition, describes the development of a Braille to speech system which uses image processing techniques. It includes translation of scanned Braille images into text by means of dynamic thresholding, adaptive Braille grid, recovery dots techniques and TTS software (Text-To-Speech) is employed for the speech part.

The first step is the thresholding which helps to filter out the wanted information part from the scanned images and discard unwanted information. After this first step, no useful information is found to be rejected and the image is ready to be processed in the pattern detection block.

The next step is used to differentiate between two sides of the double sided Braille sheet. This algorithm consists of a "shift and overlap" process since it only moves the spots downwards or upwards and carries out a logical AND.

Further, a Braille grid is created that maps the normalised distances between dots and thus specific rows and columns of Braille dots are obtained. Detected columns and rows are arranged together to create the final structure.

After the Braille dots have been successfully detected, it is analyzed and text is segmented in rows and characters. For this, the Braille mesh is used since it marks all the positions of Braille dots. Every character is converted into a binary number according to the active dots. The process consists of reading character by character and each one of the six positions that make the basic cell. Thus a set of 6 binary values (1 or 0) corresponding to each character is obtained. These binary values need only be translated to corresponding text which can be easily displayed in a monitor or notepad as a text file. The final output obtained could be either a text file or voice output through TTS software. The proposed system is fast and efficient but there are always concerns regarding real world processing capabilities.

Chapter 3

Design

3.1 System Flow

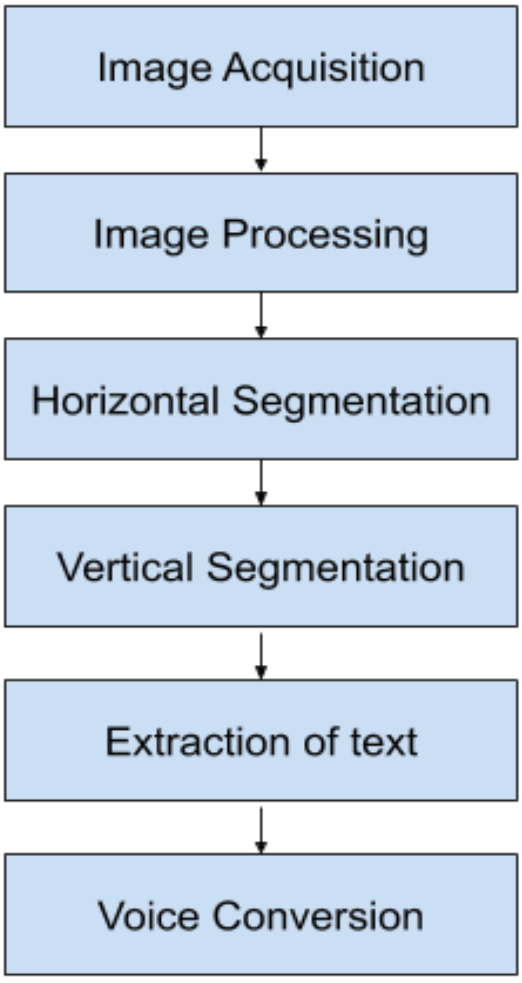


Figure 3.1: System flow

The block level representation of the project consists of 6 parts:

1. Image Acquisition
2. Image Pre processing
3. Horizontal Segmentation
4. Vertical Segmentation
5. Extraction of Text
6. Voice Conversion

3.1.1 Image Acquisition

This consists of the hardware section of the project responsible for capturing the image of the object under observation. The object under observation is a single sided braille script. Image can be captured using a camera or a scanner. Most of the researchers use the scanner as there acquisition technique, because it can solve the use of cameras problems in term of complex camera setup, low resolution, illumination problems and the misplacement of Braille dots. The use of scanner gives better results in term of Braille dots alignment and a better resolution. The scanner is attached with a single common light source, that makes the detection of Braille dots in a single side or double sided Braille papers easy.

3.1.2 Image Preprocessing

This is the software section of the project. It consists of the code required to convert the input image acquired to the required format of output. The acquired Braille images are still not perfect and it need some enhancements to be processed to the next stage in optical Braille recognition system. This enhancements can be achieved by using a sequence of image pre-processing techniques. Previous researchers focus on using a number of image processing techniques in order to enhance the acquired Braille image includes converting the image from RGB colored image to gray scale image that will make the further processing easy, because the three components of the color (Red, Green, Blue) will be compressed into one component.After converting the image into gray scale a noise removal is done using appropriate algorithms.

3.1.3 Horizontal Segmentation

Horizontal Segmentation is done in order to process each line of the document image separately. To segment the lines of a Braille text, the horizontal projection profile of the scanned Braille image is brought into being. The horizontal projection profile is the number of pixels having intensity value greater than zero along every row of the image.

3.1.4 Vertical Segmentation

Vertical Segmentation is done to achieve word segmentation. In Braille script spacing between the words is greater than the spacing between the characters in a word. This spacing between the words is found by taking the vertical projection profile of a line segmented image. Vertical projection profile is the sum of pixels having pixel intensity value greater than zero along every column of the image. From the vertical projection profile it can be observed that the width of zero-valued valleys is more between the words in a line as equated to the width of zero-valued valleys that exist between the characters in a word. This evidence is used to separate words from the line segmented input image.

3.1.5 Extraction of Text

This is done by dividing each cell into grids consisting of six parts and corresponding code for each cell is generated according to the presence or absence of a dot in each grid depending on the Braille cell placement. This helps us generate a binary code. This way of Braille text binarization makes the global system independent of the language of the document and easily configurable for adding different alphabets. Each character coded like a binary number will have to be translated to its equivalent letter in normal text to get the final output like a text file.

3.1.6 Voice Conversion

This is done with the help of gTTS package. The text file obtained is fed to the gTTS converter as a string input. It then converts the text file into its corresponding audio file. The audio file is saved in mp3 format onto a folder and is played automatically.

3.2 Block Diagram

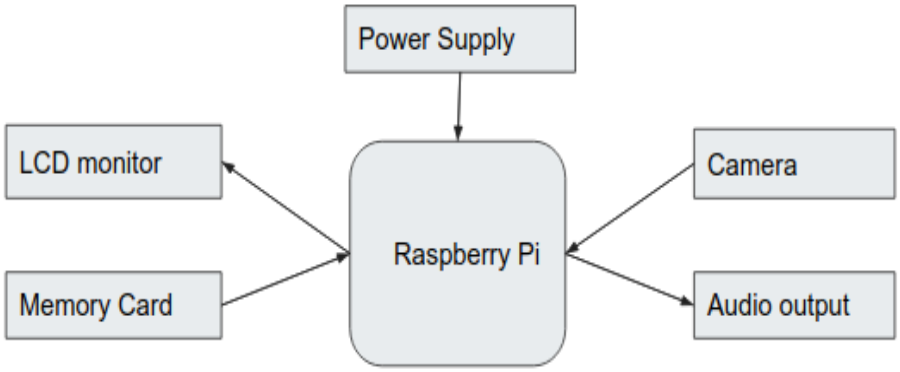


Figure 3.2: Block Diagram

3.3 System Design

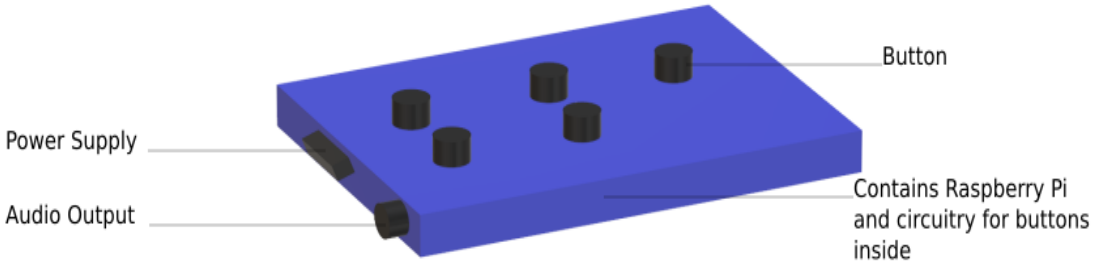


Figure 3.3: System Design

3.4 Hardware Overview

The hardware part of the project mainly consists of 3 parts:

- Image Acquisition
- Voice Output
- Keypad

Both these sections are interfaced using a Raspberry Pi.

3.4.1 Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. The Raspberry Pi was designed for the Linux operating system, and many Linux distributions now have a version optimized for the Raspberry Pi. Two of the most popular options are Raspbian, which is based on the Debian operating system, and Pidora, which is based on the Fedora operating system. Here we are using Raspbian for interfacing.

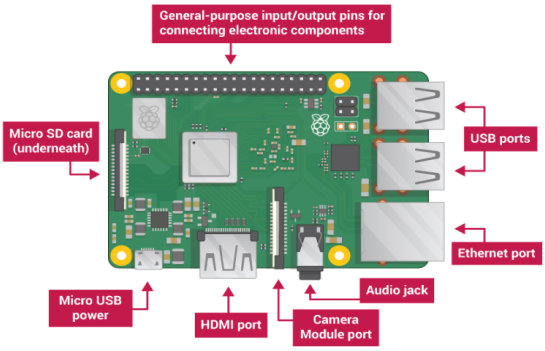


Figure 3.4: Raspberry Pi model

The main parts of a raspberry Pi include:

- SD card slot — The SD card can be slotted in here. This is where the operating system software and the files are stored.
- Ethernet port — This is used to connect the Raspberry Pi to a network with a cable. The Raspberry Pi can also connect to a network via wireless LAN.
- Audio jack — Headphones or speakers can be connected here.
- HDMI port — This is where the monitor (or projector) that is used to display the output from the Raspberry Pi is connected. If the monitor has speakers, they can be used to hear sound.
- Micro USB power connector — This is where a power supply is connected.
- GPIO ports — These allow the connection of electronic components such as LEDs and buttons to the Raspberry Pi.

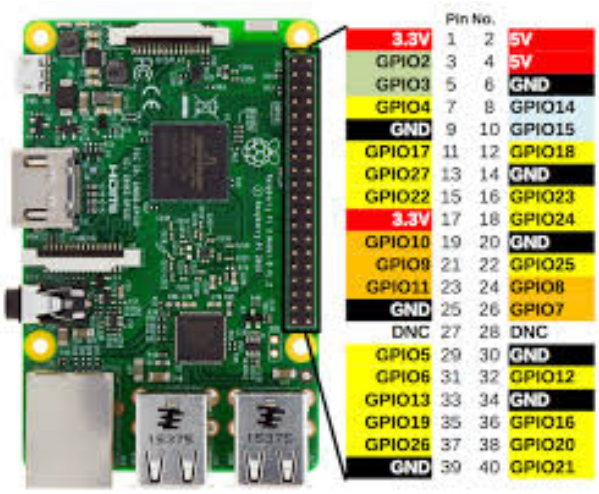


Figure 3.5: GPIO Ports

In this project, we use the Raspberry Pi 3 model.

Raspberry Pi 3 - Model B Technical Specification:

- Broadcom BCM2387 chipset
- 1.2GHz Quad-Core ARM Cortex-A53
- 802.11 bgn Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
- 1GB RAM
- 64 Bit CPU
- 4 x USB ports
- 4 pole Stereo output and Composite video port
- 10/100 BaseT Ethernet socketbr
- CSI camera port for connecting the Raspberry Pi camera
- DSI display port for connecting the Raspberry Pi touch screen display
- Micro SD port for loading your operating system and storing data
- Micro USB power source

Figure 3.6: Raspberry Pi 3 model B

Raspberry Pi 3 - Model B Features

- 10x Faster - Broadcom BCM2387 ARM Cortex-A53 Quad Core Processor powered Single Board Computer running at 1.2GHz
- 1GB RAM helps run bigger and more powerful applications
- Fully HAT compatible

- 40 pin extended GPIO to enhance real world projects.
- Connect a Raspberry Pi camera and touch screen display
- Stream and watch Hi-definition video output at 1080
- Micro SD slot for storing information and loading the operating systems.
- 10/100 BaseT Ethernet socket to quickly connect the Raspberry Pi to the Internet

3.4.2 Audio Output

Audio output is presented through either headphones or earphones, which can be connected to the audio jack, available in the Raspberry Pi.
The Raspberry Pi has two audio output modes: HDMI and headphone jack. A user can switch between these modes at any time.

If there is an HDMI monitor or TV with built-in speakers, the audio can be played over the HDMI cable, but it can easiy be switched to a set of headphones or other speakers plugged into the headphone jack. If the display claims to have speakers, sound is output via HDMI by default; if not, it is output via the headphone jack.

3.4.3 Push Button

A push-button or simply button is a simple switch mechanism for controlling some aspect of a machine or a process. Buttons are typically made out of hard material, usually plastic or metal.The surface is usually flat or shaped to accommodate the human finger or hand, so as to be easily depressed or pushed. Buttons are most often biased switches, although many un-biased buttons (due to their physical nature) still require a spring to return to their un-pushed state.

Figure 3.7: Push Button

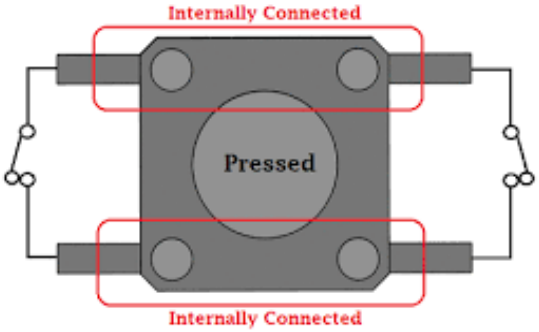


Figure 3.8: Push Button Internal Connection

Here the push buttons are used to select various options according to the requirement of the user. Six push buttons are used in this project. These have different functionalities,which are as follows:

- Button F - Move forward
- Button B - Move backward
- Button S - for processing prescanned images
- Button M - for processing real time images
- Button R - Read/Repeat

Chapter 4

Software Used

The software section of the project is the part that accepts the image input and processes it in order for it to be of the form necessary for being played as an audio file. It makes use of the following softwares.

4.1 Open CV

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, and similar images from an image database, remove red eyes from images taken using ash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

4.2 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for

use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. Python is managed by the non-profit Python Software Foundation.

Python very closely resembles the English language, using words like 'not' and 'in' to make it to where you can very often read a program, or script, aloud to someone else and not feel like you're speaking some arcane language. This is also helped by Python's very strict punctuation rules which means you don't have curly braces ( ) all over your code.

Also, Python has a set of rules, known as PEP 8, that tell every Python developer how to format their code. This means you always know where to put new lines and, more importantly, that pretty much every other Python script you pick up, whether it was written by a novice or a seasoned professional, will look very similar and be just as easy to read. The fact that a Python code, with five or so years of experience, looks very similar to the code that Guido van Rossum (the creator of Python) writes is such an ego boost.

4.3 Raspbian

Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run. However, Raspbian provides more than a pure OS: it comes with over 35,000 packages, pre-compiled software bundled in a nice format for easy installation on your Raspberry Pi.

Raspbian was created by Mike Thompson and Peter Green as an independent project. The initial build was completed in June 2012. The operating system is still under active development. Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs

.

Raspbian uses PIXEL, Pi Improved X-Window Environment, Lightweight as its main desktop environment as of the latest update. It is composed of a modified LXDE desktop environment and the Openbox stacking window manager with a new theme and few other

changes. The distribution is shipped with a copy of computer algebra program Mathematica and a version of Minecraft called Minecraft Pi as well as a lightweight version of Chromium as of the latest version.

4.4 eSpeak

eSpeak is a compact open source software speech synthesizer for English and other languages, for Linux and Windows.eSpeak uses a "formant synthesis" method. This allows many languages to be provided in a small size. The speech is clear, and can be used at high speeds, but is not as natural or smooth as larger synthesizers which are based on human speech recordings.

eSpeak is available as:

1. A command line program (Linux and Windows) to speak text from a file or from stdin.
2. A shared library version for use by other programs.
3. A SAPI5 version for Windows, so it can be used with screen-readers and other programs that support the Windows SAPI5 interface.
4. eSpeak has been ported to other platforms, including Android, Mac OSX and Solaris.

The eSpeak speech synthesizer supports several languages, however in many cases these are initial drafts and need more work to improve them. Various development tools are available for producing and tuning phoneme data.

4.5 Google Text To Speech

Google Text-to-Speech is a screen reader application developed by Google for its Android operating system. It powers applications to read aloud (speak) the text on the screen which support many languages. Text-to-Speech may be used by apps such as Google Play Books for reading books aloud, by Google Translate for reading aloud translations providing useful insight to the pronunciation of words, by Google Talkback and other spoken feedback accessibility-based applications, as well as by third-party apps. Users must install voice data for each language.Cloud Text-to-Speech is powered by WaveNet, software created by Google's UK-based AI subsidiary DeepMind. Since Google bought DeepMind in 2014, it's been exploring ways to turn the company's AI talent into tangible products. Integrating WaveNet into its cloud service is significant as Google tries to win cloud business away from Amazon and Microsoft, presenting its AI skills as its differentiating factor.

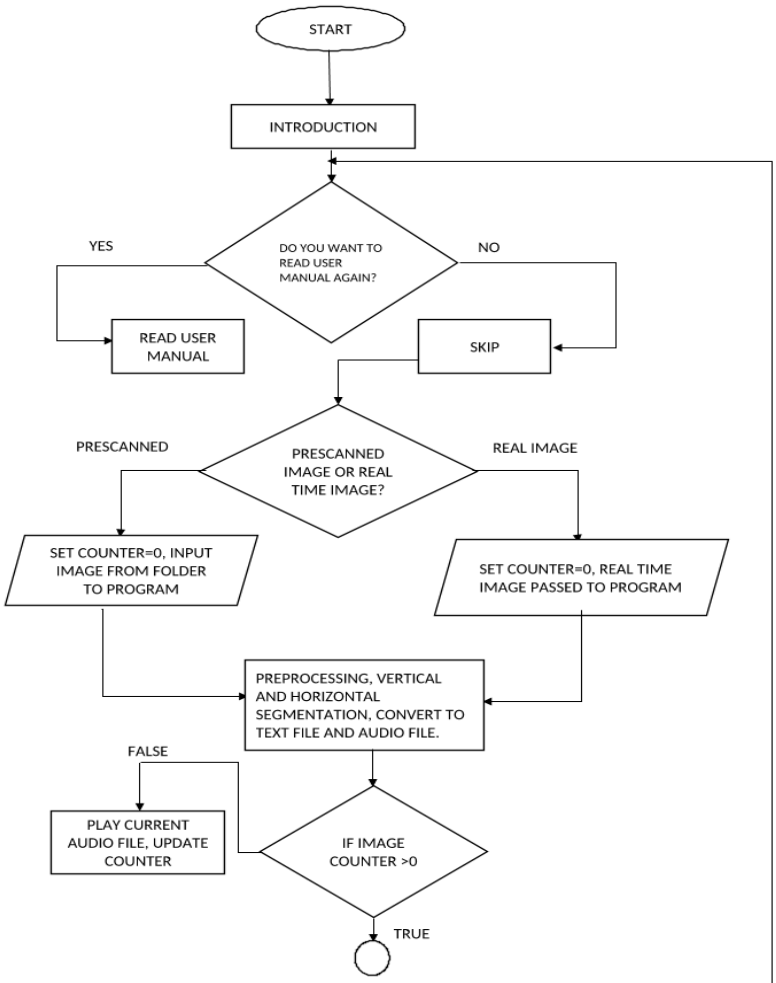
DeepMind's AI voice synthesis tech is notably advanced and realistic. Most voice synthesizers (including Apple's Siri) use concatenative synthesis, in which a program stores individual syllables — sounds such as "ba," "sht," and "oo" — and pieces them together to

form words and sentences. WaveNet instead uses machine learning to generate speech. It the waveform from a database of human speech and re-creates them at a rate of 24,000 samples per second. The end result includes voices with subtleties like lip smacks and accents. When Google first unveiled WaveNet in 2016, it was too computationally intensive to work outside of research environments, but it's since been slimmed down significantly, showing a clear pipeline from research to product.

Chapter 5

Software Description

The flowchart for the program code is as given below:



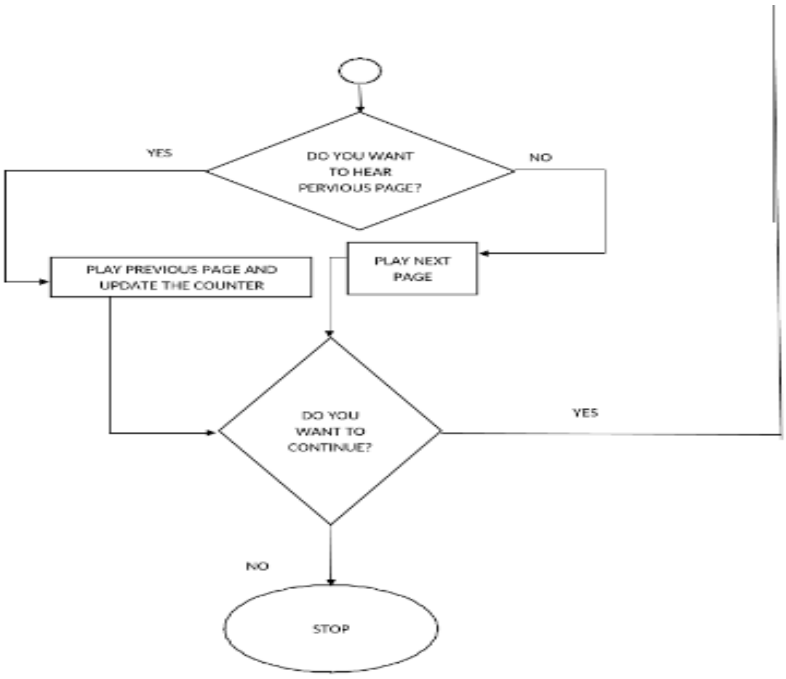


Figure 5.1: Flowchart

The program flow could be explained as:

- Start the program
- Play the introduction audio
- Check if user wishes to hear the user manual. If yes, play the user manual. If no, skip the user manual.
- Check if user wishes to process a previously scanned image from internet or process a real time scanned image.
- If user presses S, input previously scanned image from folder to program. If user presses M, input realtime image to program.
- Do the processing steps on the image sequentially, including pre-processing, horizontal and vertical segmentation, conversion to text and audio file.
- Check if image counter is 0 or greater.
- If counter is 0, play the current audio file. Update the counter.
- If counter is greater than 0, ask if user wishes to play a previous audio file or the next audio file.

- If user presses B, play previous audio file. If user presses F, process next image and play the corresponding audio file.
- Update counter.
- Repeat till no images remain.
- End the program

The 3 main software processes involved in the program are the following:

- Image Pre-processing
- Image to Text Conversion
- Text to Speech Conversion

5.1 Image Pre-processing

5.1.1 Grayscale Conversion

The RGB colour image obtained is converted into grayscale. This conversion is done mainly because it reduces the complexity of the code and increases the processing speed. With modern computers, and with parallel programming, it's possible to perform simple pixel-by-pixel processing of a megapixel image in milliseconds. Also, for many applications of image processing, color information doesn't help us identify important edges or other features.

Each picture consists of pixels that has a luminance value, regardless of its color. Luminance can also be described as brightness or intensity, which can be measured on a scale from black (zero intensity) to white (full intensity). Most image file formats support a minimum of 8-bit grayscale, which provides 512 or 256 levels of luminance per pixel. Here we have used 0-255 as the scale, 0 being black and 255 being white.

5.1.2 Binary Thresholding

This is done to convert the grayscale image into a binary image. In Binary Image, pixels are either pure black or pure white. There are no gray values in between. There are only two values (0 or 1) possibles for a pixel, this is why such images are called as Binary Images. When Gray scale image is converted into Binary Image, it uses a threshold. Suppose gray scale values are from 0 (Pure Black) to 255(Pure White) , values greater than threshold will be converted into 1 (White) and below to threshold will be converted into 0 (Black). We have used simple thresholding.

Here, the matter is straight forward. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value. The function ***cv.threshold*** is used to apply the thresholding. The first argument is the source image, which should be a grayscale image. The second argument is the threshold value which is used to classify the pixel values. The third argument is the maximum value which is assigned to pixel values exceeding the threshold. OpenCV provides different types of thresholding which is given by the fourth parameter of the function. Basic thresholding as described above is done by using the type **cv.THRESHBINARY.** The method returns two outputs. The first is the threshold that was used and the second output is the thresholded image.
The equation can be given as:
$dst(x, y) = maximumvalue, if src(x, y) > threshold$
$dst(x, y) = 0, otherwise$

5.1.3 Image Blurring

When we blur an image, we make the color transition from one side of an edge in the image to another smooth rather than sudden. The effect is to average out rapid changes in pixel intensity. The blur, or smoothing, of an image removes "outlier" pixels that may be noise in the image. Blurring is an example of applying a low-pass filter to an image. Applying a low-pass blurring filter smooths edges and removes noise from an image. Blurring is often used as a first step before we perform edge detection, or before we find the contours of an image. Larger blur kernels may remove more noise, but they will also remove detail from and image.

Blurring can be achieved by many ways. The common type of filters that are used to perform blurring are.

- Mean filter
- Weighted average filter
- Gaussian filter

Here, we have used the averaging technique for image blurring. This is done by convolving image with a normalized box filter. It simply takes the average of all the pixels under kernel area and replace the central element. This is done by the function ***cv2.blur()*** or ***cv2.boxFilter()***.

5.1.4 Morphological Dilation and Erosion

Morphology is a broad set of image processing operations that process images based on shapes. Morphological operations apply a structuring element to an input image, creating an output image of the same size. In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. By choosing the size and shape of the neighborhood, you can construct a morphological operation that is sensitive to specific shapes in the input image.

The most basic morphological operations are dilation and erosion. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the image. In the morphological dilation and erosion operations, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image. The rule used to process the pixels defines the operation as a dilation or an erosion.

- **Dilation** : The value of the output pixel is the maximum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to the value 1, the output pixel is set to 1.
- **Erosion** : The value of the output pixel is the minimum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to 0, the output pixel is set to 0.

5.2 Image to Text Conversion

5.2.1 Horizontal line and Edge Detection

Edge detection includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges. The same problem of finding discontinuities in one-dimensional signals is known as step detection and the problem of finding signal discontinuities over time is known as change detection. Edge detection is a fundamental tool in image processing, machine vision and computer vision, particularly in the areas of feature detection and feature extraction.
Here, we have made use of Canny Edge Detection algorithm, which can be done by a single function, ***cv.Canny()***.First argument is our input image. Second and third arguments are our minVal and maxVal respectively. Third argument is aperturesize. It is the size of Sobel kernel used for find image gradients. By default it is 3. Teh function methodically performs noise reduction, finding the intensity gradient of the image and does non-maximum suppression, which is followed by hysteresis thresholding.
In image processing, line detection is an algorithm that takes a collection of n edge points and finds all the lines on which these edge points lie. The most popular line detectors are the Hough transform and convolution based techniques.In image processing, line detection is an algorithm that takes a collection of n edge points and finds all the lines on which these edge points lie. The most popular line detectors are the Hough transform and convolution based techniques.

5.2.2 Hough Line Transformation

The Hough Line Transform is a transform used to detect straight lines. To apply the Transform, first an edge detection.The purpose of the technique is to find imperfect instances of

objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform. pre-processing is desirable.
We use OpenCV functions cv2.HoughLines and HoughLinesP to detect lines in an image.The Hough Line Transform is a transform used to detect straight lines.To apply the Transform, first an edge detection pre-processing is desirable.HEre, we have used Probabilistic Hough Line Transform algorithm. It is a more efficient implementation of the Hough Line Transform. It gives as output the extremes of the detected lines (x0, y0, x1, y1) In OpenCV it is implemented with the function HoughLinesP.

5.2.3 Horizontal and Vertical Segmentation

Segmentation refers to an image processing technique where input are images and output are attributes extracted from images. It subdivides the region horizontally. Segmentation accuracy determines the actual success or failure of a process. Segmentation algorithms are based on 1 to 2 basic properties of intensity values: discontinuity or similarity.
Here the approach we have used is, to detect horizontal lines in the image, which separates the Braille words into separate rows. These detected lines are drawn onto image and image is actually cut into different rows and stored as separate images. Thus horizontal segmentation of an image is completed.
Moving on to vertical segmentation, similar processes are again done on each and every row of the initial image, sequentially so that characters can be recognized in the order they appear in the Braille image. First, lines are detected in the row image vertically so as to separate each character. These lines are then drawn onto the image and an algorithm is developed by which every character is saved as a separate image.
Because there are no in-built functions to perform these steps, the program code was developed according to perform this.

5.2.4 Braille Character Recognition

A Binary pattern vector for each Braille cell is generated. A vector has a length of 6 each correspond to a dot in the Braille cell. The presence of dot is identified after counting the number of white pixels in each grid of a cell and checking whether it satisfies the threshold criterion.

'1' indicates that dot is present and '0' indicates that dot is absent in that particular position. This string of bits for the sequence of Braille alphabets is written into a file. A sequence of 6 bits are read from the file and mapped to the corresponding alphabet. If the six bits of the string are 0's, it generates a space. These alphabets are stored in a text file for further processing.

5.3 Text to Speech Conversion

The characters are recognized from the 6 bit binary equivalent assigned to characters. It is then stored in a text file. In order to convert these characters into speech, GTTS package is used. The text that is to be converted into speech is fed into the GTTS in the form of a string input. Strings of information are then converted to speech.

Chapter 6

System Implementation Results

6.1 Phase 1

The acquired image from internet was first converted into a grayscaled image to which binary thresholding was applied to procure a simple black and white image, which would be used from then on for further processing. Here, normal binary thresholding function was used as that was found to be the best method to get the image in the required form.

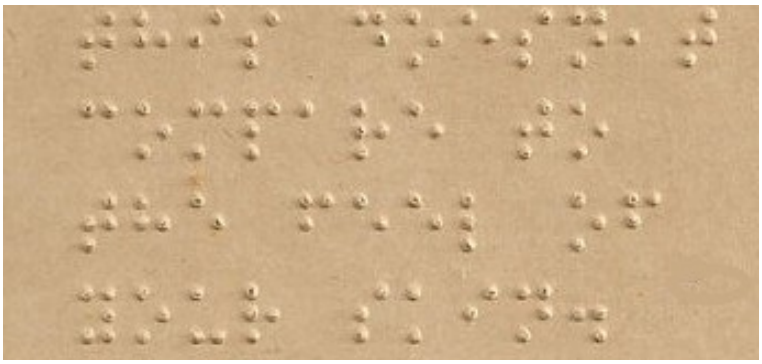


Figure 6.1: Acquired image

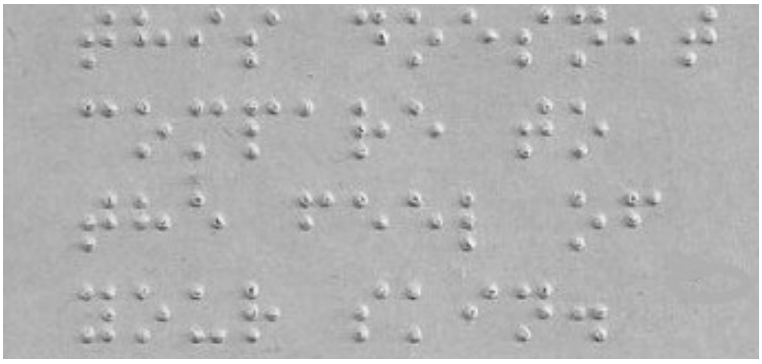


Figure 6.2: Grayscaled image

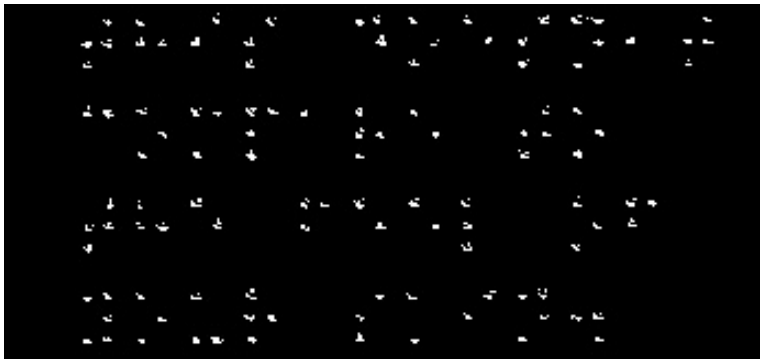


Figure 6.3: Binary thresholded image

Figure 6.1 shows the acquired image.
Figure 6.2 shows the grayscaled image.
Figure 6.3 shows the binary thresholded image.

Because binary thresholding does not provide a sufficiently clear and noise free image, morphological operations like image blurring, erosion and dilation are performed on the thresholded image. The dilated image would be used for further processing after noise has been removed. The reason why a dilation is required after performing erosion is because once erosion is performed on an image, the required areas of image for processing might become blurred or undetectable whereas dilation helps in enlarging these required areas.

Figure 6.4: Blurred image

Figure 6.5: Eroded image

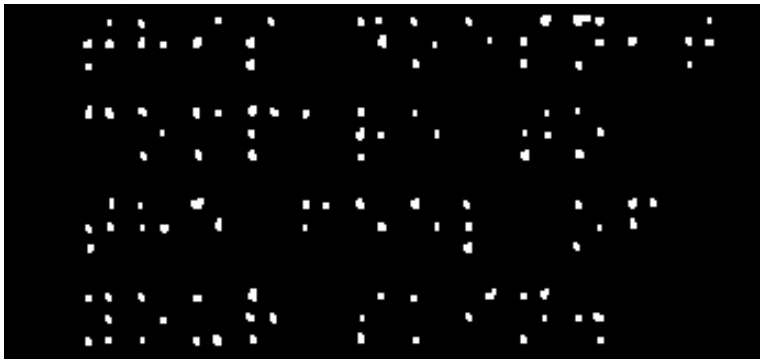


Figure 6.6: Dilated image

Figure 6.4 shows the blurred image.
Figure 6.5 shows the eroded image.
Figure 6.6 shows the dilated image.

After a proper noise-free image was obtained, horizontal and vertical segmentation processes were performed to divide the image into rows and then further into characters. First, horizontal line detection using structural element was done on the dilated image to recognize the horizontal lines and canny edge detection was used to detected the edges of lines. Hough line transform was used to draw lines separating each row.

Figure 6.7 shows the horizontal detection of lines. These lines divide the image into different rows.
Figure 6.8 shows the detection of edges of horizontal lines detected in the previous image.
Figure 6.9 shows the image after applying hough line transformation.

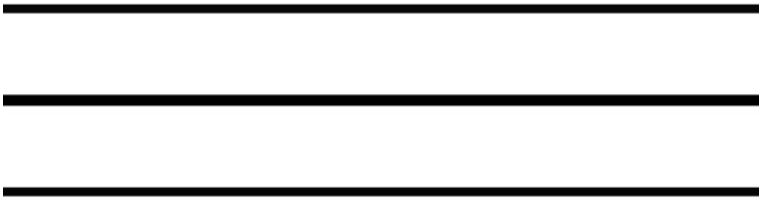


Figure 6.7: Horizontal line detection

Figure 6.8: Horizontal edge detection

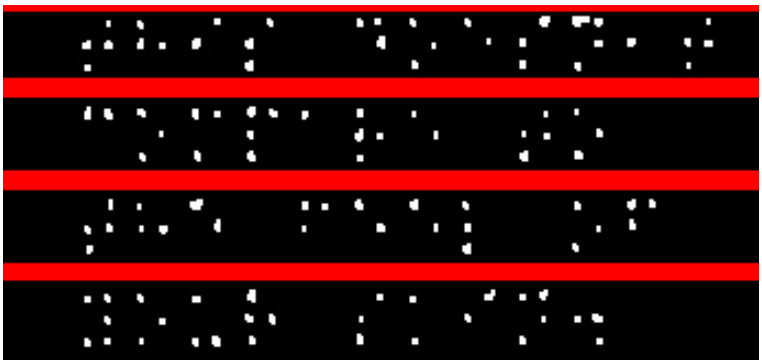


Figure 6.9: Hough line transformation

Figures 6.10, 6.11 and 6.12 show a few of the different rows obtained. Each row contains a number of Braille words, which will further be separated into characters.

Figure 6.10: Row 1

Figure 6.11: Row 2

Figure 6.12: Row 3

The same procedures as followed for horizontal segmentation were followed to detect vertical lines, their edges and again, hough line transform was used to segment characters in each row into individual images.

Figure 6.13 shows the vertical detection of lines. These lines divide the image into different characters.

Figure 6.13: Vertical line detection

Figure 6.14 shows the detection of edges of vertical lines detected in the previous image.

Figure 6.14: Vertical edge detection

Figure 6.15 shows the image after applying hough line transformation.

Figure 6.15: Vertical hough line transformation

Figures 6.16, 6.17, 6.18 shows some of the different characters obtained after vertical segmentation.

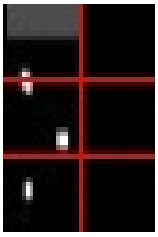


Figure 6.16: Character 1

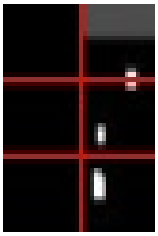


Figure 6.17: Character 2

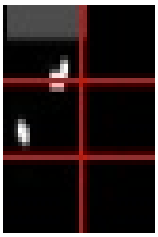


Figure 6.18: Character 3

After obtaining the separated Braille characters, the next step was the conversion of these characters into text. For this, every character was first assigned an equivalent 6-bit binary value, as set by the presence of active dots in every cell. This implies that, for every character, if there is a dot in one of the 6 cells depicting the character, it would translate to a '1' in binary. If no dot is present in a cell, it means a '0' in binary.
By mapping each of the unique binary values to English alphabets, equivalent English text of the scanned Braille text was obtained. Using this text output, voice output was obtained via GTTS module. Figure 6.19 shows theequivalent binary values of different English alphabets.

The main problems encountered in this phase were that though horizontal segmentation was somewhat a success, the program failed to split the final rows in the image into separate images perfectly, due to which further processing was adversely affected.

In vertical segmentation, the code was not compatible enough to properly recognize and accurately split each character into separate images. Further, the line division to indicate the six different dot cells in each character was also flawed. Some of the errors observed were the presence of two dots in one cell and the presence of a dot in the middle of partition of cells, both of which leads to ambiguous text detection.

The result of these issues was that a proper text output could not be obtained in the initial phase. The obtained output was riddled with errors which progressively increased in each working loop of the program. Figure 6.20 shows the obtained text output., when ideally it should have been as shown in figure 6.21.

Equivalent Binary Values of Characters			
a	100000	n	101110
b	110000	o	101010
c	100100	p	111100
d	100110	q	111110
e	100010	r	111010
f	110100	s	011100
g	110110	t	011110
h	110010	u	101001
i	010100	v	111001
j	010110	w	010111
k	101000	x	101101
l	111000	y	101111
m	101100	z	101011

Figure 6.19: Equivalent binary values of characters

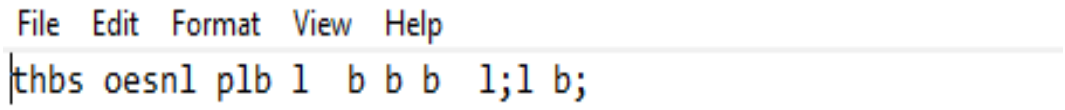


Figure 6.20: Text output - Obtained

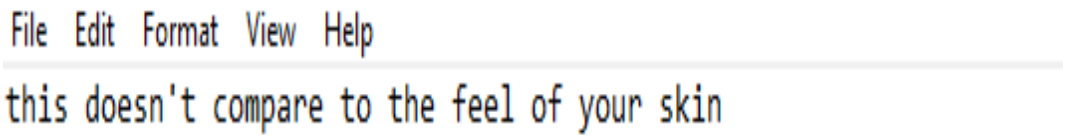


Figure 6.21: Text output - Ideal

6.2 Phase 2

This phase focused upon improving the program so that horizontal and vertical segmentation could be done more accurately. The program code optimization was mainly divided into four parts:

- Pre-processing
- Horizontal Segmentation
- Vertical Segmentation

- Text to voice conversion

The main objective was to optimise the code to process different images, without errors. Initially, different images were fed into the program, to check whether pre-processing was being done properly on different images. The following are some of the test images that were used.

Figure 6.22 has only a single row of text, which has been processed correctly as shown in figure 6.23.

Figure 6.22: Test Image 1

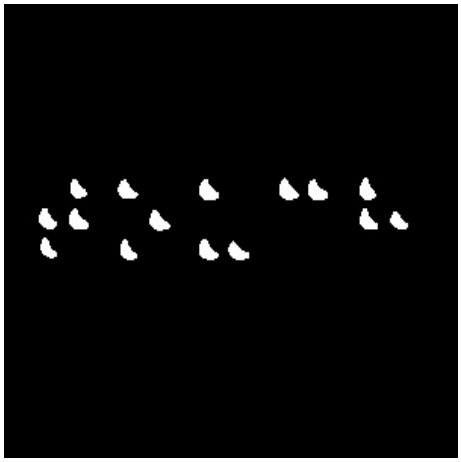


Figure 6.23: Test Image 1 after processing

Figure 6.24 has multiple lines of text, and as shown here in figure 6.25 pre-processing and de-noising has provided a much clearer image for segmentation.

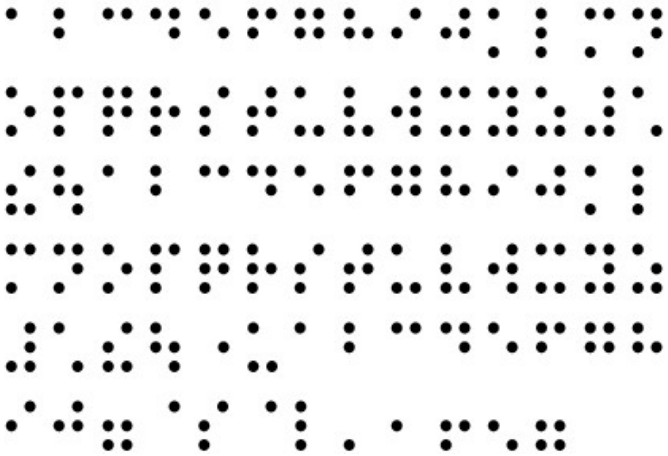


Figure 6.24: Test Image 2

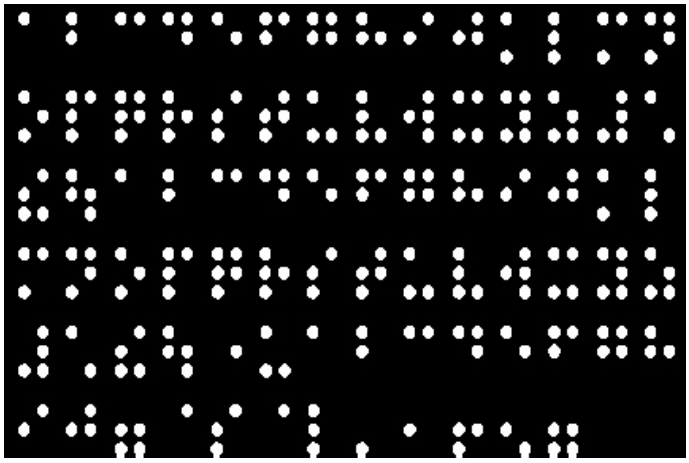


Figure 6.25: Test Image 2 after processing

Here two of the errors detected in the first phase were corrected, namely: Horizontal segmentation done perfectly, for different images. The program code was modified so that vertical segmentation was improved.
There were still problems with vertical segmentation, which was while dividing each character into 6 dot cells, some of the dots were being overlapped by lines drawn, which led to ambiguity during text detection. This is demonstrated in figure 6.26

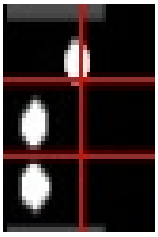


Figure 6.26: Error of ambiguity

The obtained text output is shown in figure 6.31

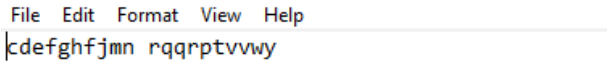


Figure 6.27: Obtained text output

6.3 Phase 3

After phase two the developed program was able to do pre-processing, de-noising and horizontal segmentation without errors. Phase 3 focused on perfecting vertical segmentation, precise detection of braille characters into corresponding text and voice conversion.
The vertical segmentation code was altered so that the errors encountered in the previous two phases were removed. The program code, after modifications was successfully able to recognize 22 out of 26 English alphabets without errors. Different test vectors were introduced to the program to check the efficiency,which was found to be around 85 percent.

The following are images from few of the test images provided.

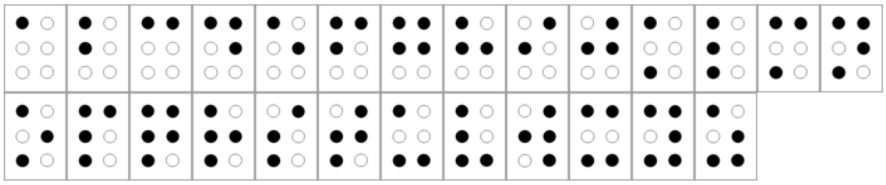


Figure 6.28: Test Image 3

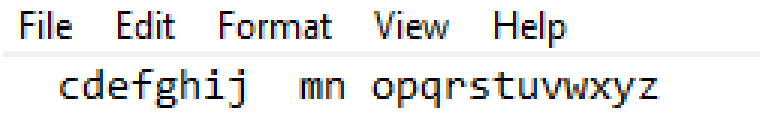
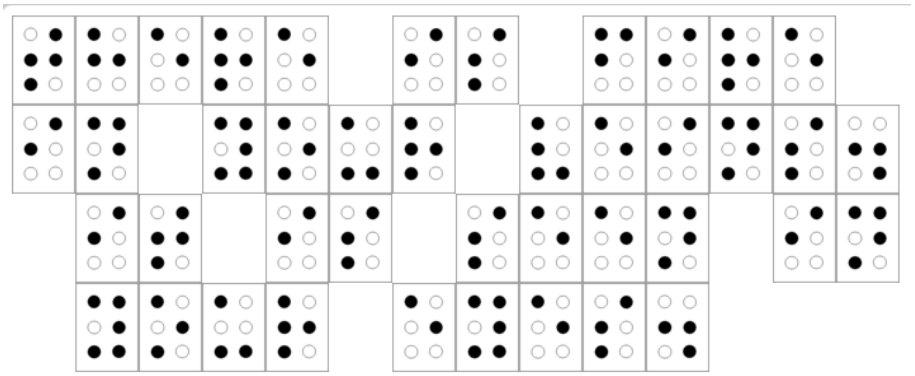


Figure 6.29: Obtained text output



in vein.png

Figure 6.30: Test Image 4

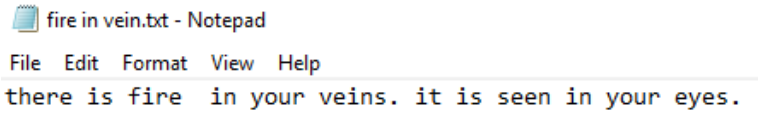


Figure 6.31: Obtained text output

6.4 Phase 4

Phase 4 focused on expanding the current classifier set, to recognize numbers as well as certain commonly used special characters, like comma, full stop etc. The program was also introduced with real time captured images to check if it worked on those images.
Real time images were captured with
a) 12 MP + 5 MP Mobile camera, with an image resolution of 4000 x 3000 pixels.
b) 18 MP Digital camera - Canon 1300 D with 9,20,000 dots resolution.
Few of the images captured are shown below:

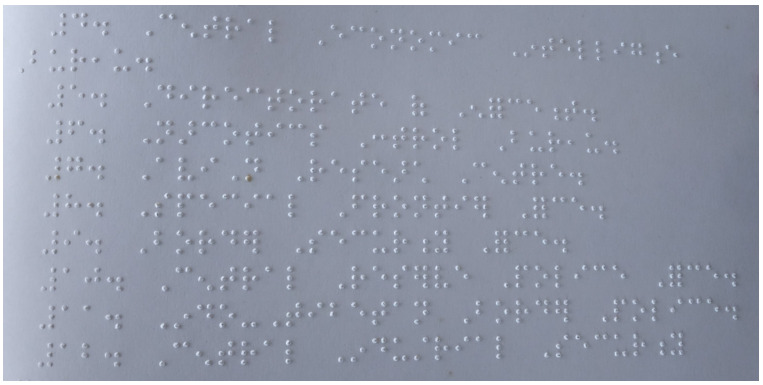


Figure 6.32: Real Time Captured Image 1

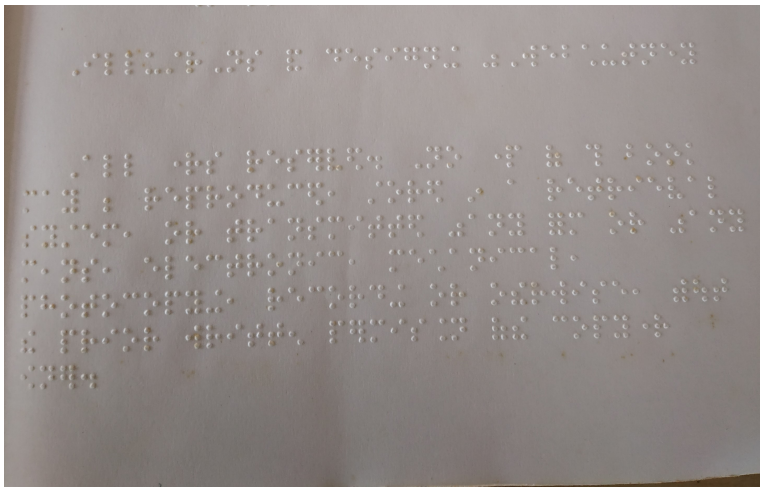


Figure 6.33: Real Time Captured Image 2

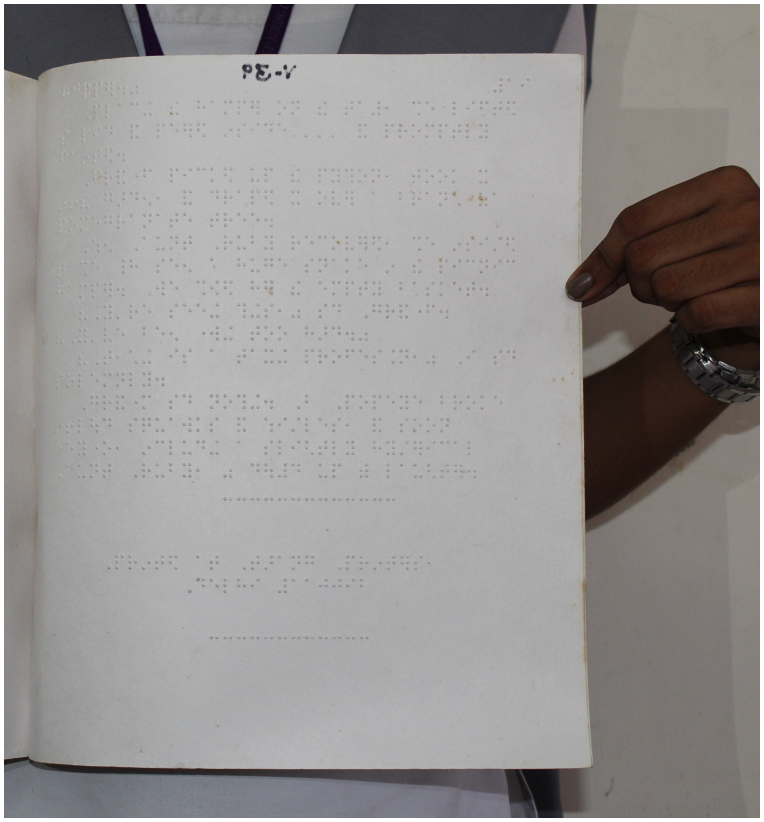


Figure 6.34: Real Time Captured Image 3

The main difficulty encountered was decoding the whole image in one go, as it was of a lower resolution than the images obtained online. This was solved by incorporating a cropping function which allowed the processing of a specific region in the image.
It was found that most of the letters were being properly recognized with only slight errors, where the character was mistakenly recognized as another. Mostly, the character was misrepresented as the letter 'l' because in the Braille standard, it is the one with most number

of white dots.

After analysing the problems, it was concluded that the program performed character recognition to the best of its ability and was compatible with real time images as well; and voice conversion was successfully performed.

6.5 Phase 5

This phase was about interfacing the entire code into Raspberry Pi and checking if the code successfully worked in it with proper outputs.
The following were the steps performed:

- Installing Rasbian OS and initializing Raspberry Pi.

Figure 6.35: Initialising Raspberry Pi

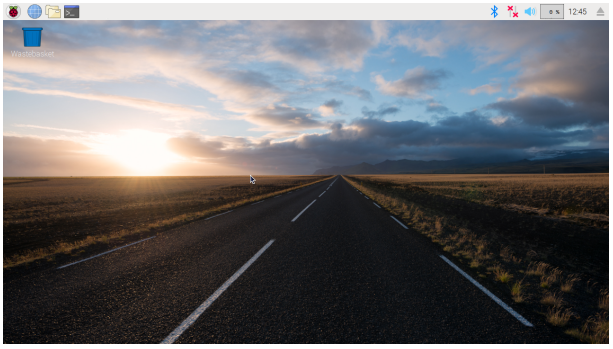


Figure 6.36: Screenshot of Raspberry Pi Desktop

- Installing the required libraries to run the program code in Raspberry Pi.

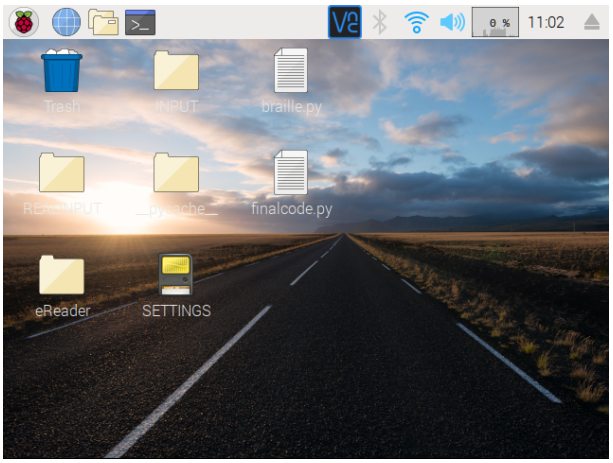


Figure 6.37: Screenshot of Raspberry Pi Desktop after installations

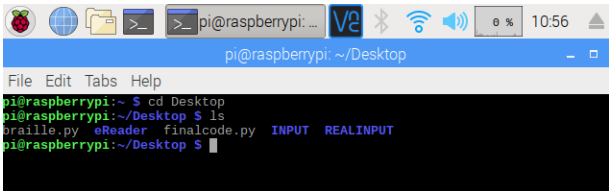


Figure 6.38: Screenshot of Raspberry Pi Terminal after installations

- Integration of push buttons into the code.

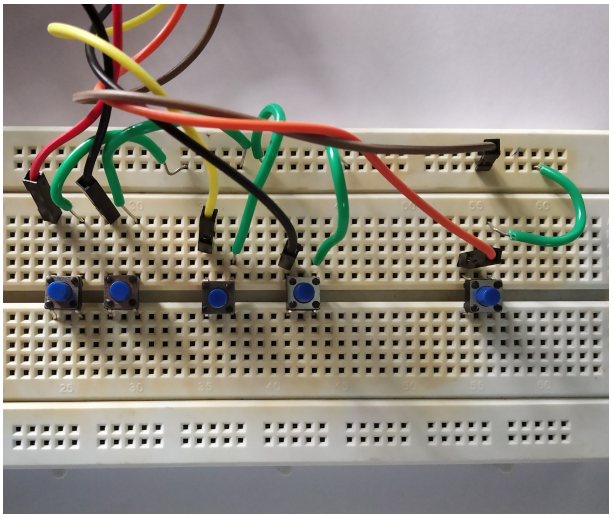


Figure 6.39: Testing the push buttons

- Testing the final code.

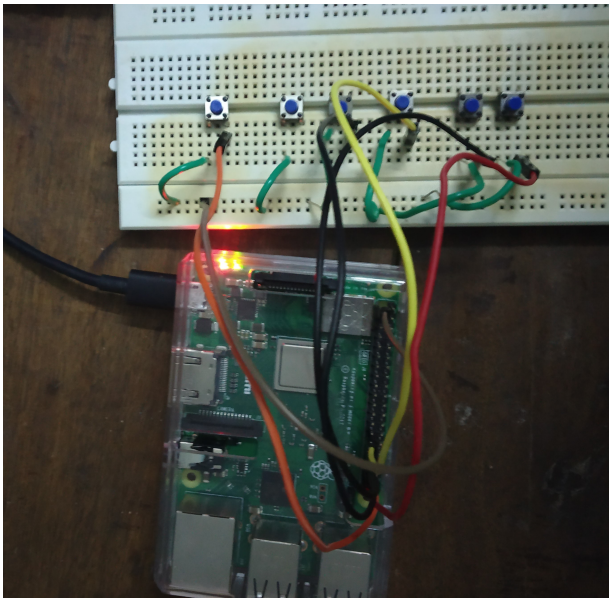


Figure 6.40: During the testing of code on Raspberry Pi

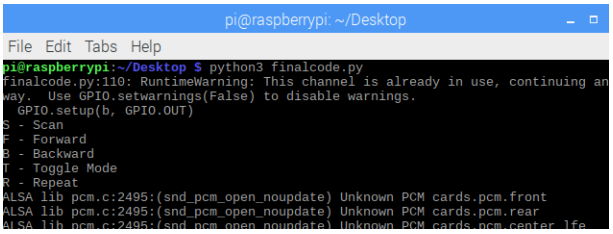


Figure 6.41: Running the code on Raspberry Pi

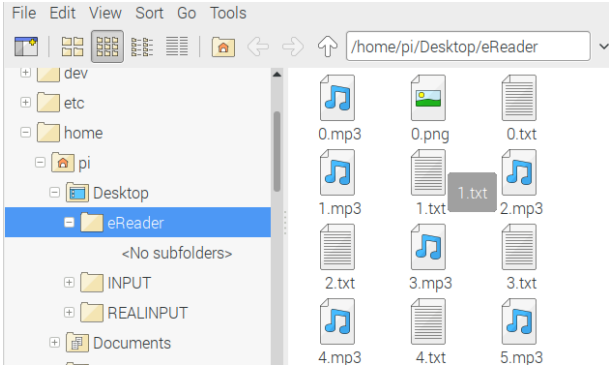


Figure 6.42: After execution of code

As understood from the figures above, the code was successfully integrated into Raspberry Pi and it is clearly shown that the program works successflly, with the results being saved as depicted in the figure 6.42. The program uses eSpeak to automatically play the saved mp3 files.

Chapter 7

Conclusion and future scope

eReader is a system developed to reduce the gap between blind, visually impaired people and sighted people; the system is able to convert the Braille characters into its corresponding natural language character, which in the case of this project is English. It can help anyone who has to interact with blind people and don't know Braille scripting language. Our system converts the Braille characters into the subsequent English language character. It is affordable and does not need expensive equipment to work.

We performed a set of sequentially designed engineering steps to produce a readable text out of the Braille document. It includes image pre-processing, noise reduction, image enhancement, character segmentation, dot extraction and character recognition. Our system's recognition ability ranges between 85% to 97%.

The future scope of this project is vast as it can cater to the demands of visually impaired and sighted people equally. Few of the future modifications that could be incorporated into the system include:

- Incorporating a monitor compatible with Raspberry Pi, which would also be portable. This would help the sighted people to just read the text that would be displayed on the screen as an alternative to using headphones/speakers to hear.
- Expanding the area of operation of the code to decode double sided braille and include conversion of other languages, which may include some of the regional as well as national languages.
- Creating a method by which the stored audio or text files or both may be stored in cloud for later retrieval, and thus creating a vast digital library that can store the Braille books and which would be universally accessible.

We strongly believe that our system has a great potential for daily use to enhance communication with this important and effective population in each society.

Bibliography

[1] PYN De Silva, N. Wedasinghe, "Braille Converter and Text-To-Speech Translator for Visually Impaired People in Sri Lanka",*American Journal of Mobile Systems, Applications and Services*, Vol. 3, No. 1, 2017, pp. 1-9

[2] 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",*International Conference on Intelligent Computing,Instrumentation and Control Technologies (ICICICT)*,2017

[3] Fatih Erden, "Literary Braille Language Translator to Spanish Text",*American Journal of Mobile Systems, Applications and Services*, Vol. 3, No. 1, 2017, pp. 1-9

[4] S.Padmavathi, Manojna K.S, Sphoorthy Reddy and Meenakshy."Conversion of Braille to Text in English, Hindi and Tamil Languages",*International Journal of Computer Science, Engineering and Applications (IJCSEA) Vol.3, No.3*,June 2013

[5] Li Nian-feng,Wang Li-rong, "A kind of Braille paper automatic marking system", *2011 International Conference on Mechatronic Science, Electric Engineering and Computer*August 19-22, 2011, Jilin, China

[6] Suárez Molina, Barrera Pérez and Jacinto Gómez,"Image Processing Techniques for Braille Writing Recognition", *IEEE transaction,9781509011476/1631*,2016

[7] S. Halder, A. Hasnat, A. Khatun, D. Bhattacharjee, and M. Nasipuri, "Development of a bangla character recognition (bcr) system for generation of bengali text from braille notation," *International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN*, pp. 2278–3075.

Appendices

Appendix A

Program Code

```
1 #!/usr/bin/python
2
3 from __future__ import print_function
4 import argparse
5 import math
6 import string
7 from gtts import gTTS
8 import cv2
9 import numpy as np
10 from PIL import Image
11 import time
12 import datetime
13 from array import array
14 import sys
15 import os
16 import subprocess
17 import PIL.Image as Image
18 from glob import glob
19 from braille import realtime
20 import RPi.GPIO as GPIO
21 import ctypes
22 mpDir = "/home/pi/Desktop/"
23 ####################
24 #shutdown function#
25 ####################
26 def my_safeShutdown():
27   print("Program startup.")
28   counter = 0
29   exit = False
30   counterFile = open( mpDir + "mp_sdCounter.txt", "r")
31   if(counterFile.closed == False):
32     counter = int(counterFile.read()) #count number of contents in counterFile
33     print("shut down counter = ", counter)
34     if(counter <= 0):
35       print("shut down counter is less than 1. Exiting.")
36       return
37     counterFile.close()
38     text = subprocess.check_output("sudo shutdown -h 2", shell = True) #
```

```
    shutdown system after 2 min
39      print("\n" + text.decode("utf-8"))
40      counter = counter - 1
41      counterFile = open( mpDir +"mp_sdCounter.txt", "w")
42      counterFile.write(str(counter))
43      counterFile.close()
44      print("\nSuccess of my_safeShutdown().")
45    else:
46      print("No counter file.")
47 now = datetime.datetime.now()
48 #all these are GPIO PIN NUMBERS NOT PHYSICAL PIN NUMBERS
49 #assigning variables to buttons
50 button_F = 26 #forward
51 button_B = 6  #backward
52 button_S = 13 #prescanned image
53 button_M = 5  #realtime image
54 button_R = 19 #repeat/read
55 #flags
56 button_MULTIPLE = -1
57 button_NOTSET = 0
58 button_SHUTDOWN = -3
59 buttons = [button_S, button_M, button_B, button_R, button_F]
60 ######################
61 # BUTTON Functions#
62 ######################
63 #set up button as output, initialise o/p to 0
64 def my_setupInput():
65     GPIO.setmode(GPIO.BCM)
66     for b in buttons:
67         GPIO.setup(b, GPIO.OUT)
68         GPIO.output(b, 0)
69 #find out which button was pressed. (the pin number)
70 def my_InputButton():
71     pressedButton = button_NOTSET
72     i = 0
73     for b in buttons:
74         if(GPIO.input(b) == True):
75             pressedButton = pressedButton | (1 << i)
76         i = i + 1
77     return pressedButton
78 #define input button acc. to pin number of button pressed.
79 def my_Input():
80     pressedButton = button_NOTSET
81     while(pressedButton == button_NOTSET):
82         pressedButton = my_InputButton()
83     buttonsWhichWerePressed = button_NOTSET
84     while(pressedButton == button_NOTSET):
85         pressedButton = my_InputButton()
86     buttonsWhichWerePressed = buttonsWhichWerePressed | pressedButton
87     while(pressedButton != button_NOTSET):
88         pressedButton = my_InputButton()
89         buttonsWhichWerePressed = buttonsWhichWerePressed | pressedButton
90
91     pressedButton = buttonsWhichWerePressed
```

```
92      print("pressedButton = ", pressedButton)
93      if(pressedButton == 6):
94          return 'E'
95      elif(pressedButton == 32):
96          return 'F'
97      elif(pressedButton == 2):
98          return 'S'
99      elif(pressedButton == 8):
100         return 'B'
101     elif(pressedButton == 4):
102         return 'M'
103     elif(pressedButton == 1):
104         return 'T'
105     elif(pressedButton == 16):
106         return 'R'
107     return 'X'
108 ######################################
109 #preprocess and horizontal function#
110 ######################################
111 def processImage(img):
112         #PART 1
113         #Invert the image
114         img = 255 - img
115         ret, thresh = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
116         blur = cv2.blur(thresh,(5,5))
117         kernel = np.ones((5,5),np.uint8)
118         erosion = cv2.erode(blur,kernel,iterations = 1)
119         ret, thresh2 = cv2.threshold(erosion, 12, 255, cv2.THRESH_BINARY)
120         kernel = np.ones((3,2),np.uint8)
121         mask = cv2.dilate(thresh2,kernel,iterations = 1)
122         rows,cols=mask.shape
123         cv2.imwrite('cmask.jpg',mask)
124         cv2.destroyAllWindows()
125         #PART 2
126         th2=cv2.imread('cmask.jpg')
127         r,c,w=mask.shape
128         horizontalStructure = cv2.getStructuringElement(cv2.MORPH_RECT, (c
    +2000,13))
129         horizontal = cv2.dilate(th2, horizontalStructure, iterations = 1)
130         #defining the edges
131         edges = cv2.Canny(horizontal,50,150,apertureSize = 3)
132         #finding the end points of the hough lines
133         lines = cv2.HoughLines(edges,1,np.pi/180,200)
134         m=[]
135         minLineLength = 100
136         maxLineGap = 10
137         lines = cv2.HoughLinesP(edges,1,np.pi/180,15,minLineLength,maxLineGap)
138         #lines contain values of r and theta so here we have n such values
139         for x in range(0, len(lines)):
140                 #print ("lines array:",lines[x])
141                 for x1,y1,x2,y2 in lines[x]:
142                         m.append(((x1,y1),(x2,y2)))
143         #printed m, it has (x1,y1),(x2,y2) for each of the n such iteration
    grouped.
```

```
144         sorted_m=sorted(m, key=lambda x: x[0][1])
145         #printed(sorted_m) it contains list m that is sorted using
146         #lambda (anonymous fn) which takes argument x, which refers to
        elements
147         #in the list m.
148         #in every iteration, it takes element in index [0][1] and sorts.
149         sorted_m.insert(0,((0,0),(c,0)))
150         #now sorted list has extra element (0,0),(430,0) so size of sorted m
        is now 11
151         #drawing line
152         for i in range (0,len(sorted_m)):
153             cv2.line(th2,sorted_m[i][0],sorted_m[i][1],(0,0,255),1)
154         p=[]
155         for i in range (0,len(sorted_m)):
156             if i!=len(sorted_m)-1:
157                     p.append(th2[sorted_m[i][0][1]:sorted_m[i+1][0][1],
        sorted_m[i][0][0]:sorted_m[i][1][0]])
158 
159             else:
160                     p.append(th2[sorted_m[len(lines)-1][0][1]:sorted_m[len(
        lines)][0][1], sorted_m[len(lines)][0][0]:sorted_m[len(lines)][1][0]])
161         pix=[]
162         for x in range(len(p)):
163                 #down below is a function called contains_white with argument
        img, it is grayscaled thresholded, found shape.
164             def contains_white(img):
165                 gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
166                 ret, threshold = cv2.threshold(gray_image,100,255,cv2.
        THRESH_BINARY)
167                 h,w,l=img.shape
168                 # for loop with range as rows of img, second for loop with
        range as coumns of image, find areas of white
169                 for i in range(h):
170                     for j in range(w):
171                         if threshold[i][j]==255:
172                             return True
173             result= contains_white(p[x])
174             if result== True:
175                 pix.append(p[x])
176         x=len(pix)
177         for i in range(len(pix)):
178                 cv2.imwrite('part'+str(i)+'.jpg',pix[i])
179         cv2.destroyAllWindows()
180         return x
181 ####################################
182 #vertical + brailletotext function#
183 ####################################
184 def VertSeg(f,x):
185         for e in range(x-1):
186             j=str(e)
187             img=cv2.imread('part'+j+'.jpg',0)
188             rows,cols= img.shape
189             ret, threshold = cv2.threshold(img,200,255,cv2.THRESH_BINARY)
190             #dilating
```

```
191             kernel = np.ones((100,10),np.uint8)
192             dilation = cv2.dilate(threshold,kernel,iterations = 1)
193             #defining the edges
194             edges = cv2.Canny(dilation,50,150,apertureSize = 3)
195             m=[]
196             minLineLength = 100
197             maxLineGap = 10
198             lines = cv2.HoughLinesP(edges,1,np.pi/180,15,minLineLength,
    maxLineGap)
199             for x in range(0, len(lines)):
200                 for x1,y1,x2,y2 in lines[x]:
201                     m.append(((x1,y1),(x2,y2)))
202             #print('m is:',m) ----> list of elements as [(x,y),(x,y)...]
203             sorted_m=sorted(m, key=lambda x: x[0][0])
204             #print('sorted m is:',sorted_m) ---> sorted as per (0,0)th element
205             #drawing lines
206             for i in range(len(sorted_m)):
207                 cv2.line(img,sorted_m[i][0],sorted_m[i][1],(0,0,255),1)
208             cv2.imwrite('houghPart'+j+'.jpg',img)
209             #defining function distance
210             def distance(f,g):
211                 if f==sorted_m[-1][0]:
212                     return 0
213                 else:
214                     dx=g[0]-f[0]
215                     dy=g[1]-f[1]
216                     d =math.sqrt(dx*dx+dy*dy)
217                     return d
218
219             def dis(f,g):
220                 dx=g[0]-f[0]
221                 dy=g[1]-f[1]
222                 d =math.sqrt(dx*dx+dy*dy)
223                 return d
224
225             hough_lines=cv2.imread('houghPart'+j+'.jpg')
226             p=[]
227             sorted_m.append(((cols,rows),(cols,0)))
228             #saving each character in list p
229             for i in range(len(sorted_m)-1):
230                 p.append(hough_lines[sorted_m[i][1][1]:sorted_m[i][0][1],
    sorted_m[i][0][0]: sorted_m[i+1][0][0]])
231             pxl=[]
232             r=0
233             def contains_space(img):
234                 ret, threshold = cv2.threshold(img,200,255,cv2.THRESH_BINARY)
235                 h,w,l=img.shape
236                 for g in range(h):
237                     for h in range(w):
238                         a = threshold[g][h]
239                         if a.all() ==255:
240                             return 1
241                 return 0
242             #removing the unnecesary space
```

```
243             dots=[]
244             o=[]
245             for i in range(len(p)):
246                 j=p[i]
247                 if contains_space(j)==1:
248                     pxl.append(p[i])
249                     dots.append(np.array(p[i]))
250                     o.append(p[i].shape[1])
251                 elif contains_space(j)==0:
252                     if p[i].shape[1]>20:
253                         pxl.append(p[i])
254                         dots.append(np.array(p[i]))
255                         o.append(p[i].shape[1])
256 
257                 else:
258                     pass
259 
260             def position(img):
261                 pos=[]
262                 ret, threshold = cv2.threshold(img,200,255,cv2.THRESH_BINARY)
263                 h,w,l=img.shape
264                 for i in range(h):
265                     for j in range(w):
266                         a = threshold[g][h]
267                         if a.all()==255:
268                             pos.append((j,i))
269                 v=0
270                 for i in range(len(pos)-1):
271                     if (pos[i+1][0]-pos[i][0]) >5:
272                         return False
273                 return True
274 
275             #checking if p[i] ==img then return the width of adjecent space
    for right and left of img
276             def xyz_r(img):
277                 for i in range(len(p)-1):
278                     if img.shape == p[i].shape and not(np.bitwise_xor(img,p[i
    ]).any()):
279                         #print(np.bitwise_xor(img,p[i]).any())
280                         return p[i+1].shape[1]
281 
282             def xyz_l(img):
283                 for i in range(len(p)):
284                     if img.shape == p[i].shape and not(np.bitwise_xor(img,p[i
    ]).any()):
285                         return p[i-1].shape[1]
286             no_of_dots=[]
287             j=str(dots)
288             for i in range(len(pxl)):
289                 c=0
290                 imgray=cv2.imread(j,0)
291                 ret,thresh = cv2.threshold(imgray,147,255,cv2.THRESH_BINARY)
292                 image, contours, hierarchy = cv2.findContours(thresh,cv2.
    RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

```
293                 for cnt in contours:
294                     colour=(34,0,255)
295                     thick=10
296                     cv2.drawContours(dots[i],[cnt],0,colour,thick)
297                     c=c+1
298                 no_of_dots.append(c)
299 
300             zs={}
301             for i in range(len(pxl)):
302                 zs[str(i)]=[]
303             for i in range(len(pxl)):
304                 if pxl[i].shape[1] in range(min(o),(sum(o)//len(o))):
    # min(o),(sum(o)//len(o) is aroun the range 31, 43
305                     def contains_whitepx(img,x):
306                         ret, threshold = cv2.threshold(img,200,255,cv2.
    THRESH_BINARY)
307                         h,w,l=img.shape
308                         for g in range(h):
309                             for h in range(w):
310                                 a = threshold[g][h]
311                                 if a.all()==255:
312                                     zs[str(x)].append((h,g))
313                         return 0
314                     # print('i is', i,pxl[i].shape[1] ) # 1, 39.... 2,39.....
    3,38.... 11,38
315                     contains_whitepx(pxl[i],i)
316                     if len(zs[str(i)]) in range(0,20):
317                         RD = xyz_r(pxl[i])
318                         LD = xyz_l(pxl[i])
319                         h1, w1 = pxl[i].shape[:2]
320                         black_image = np.zeros((pxl[i].shape[0], 42-pxl[i].
    shape[1],3))
321                         h2, w2 = black_image.shape[:2]
322 
323                         #create empty matrix
324                         vis = np.zeros((max(h1, h2), w1+w2,3), np.uint8)
325                         if RD != None and LD != None:
326                                 if RD>LD:
327                         #combine 2 images
328                                         vis[:h1, :w1,:3] = pxl[i]
329                                         vis[:h2, w1:w1+w2,:3] = black_image
330                                         resized_image=cv2.resize(vis,(40,40))
331                                         pxl[i]=resized_image
332                                         rows,cols,w=pxl[i].shape
333                                         for x in range(rows):
334                                             for y in range(cols):
335                                                 pixel=pxl[i][x][y]
336                                                 if pixel[2]>pixel[1] and pixel
    [2]>pixel[0]:
337                                                     pxl[i][x][y]=[0,0,0]
338 
339                                 elif LD>RD:
340                                         vis[:h2, :w2,:3] = black_image
341                                         vis[:h1, w2:w1+w2,:3] = pxl[i]
```

```
342                                         resized_image=cv2.resize(vis,(40,40))
343                                         pxl[i]=resized_image
344                                         rows,cols,w=pxl[i].shape
345                                         for x in range(rows):
346                                             for y in range(cols):
347                                                 pixel=pxl[i][x][y]
348                                                 if pixel[2]>pixel[1] and pixel
    [2]>pixel[0]:
349                                                     pxl[i][x][y]=[0,0,0]
350
351                                 else:
352                                         resized_image=cv2.resize(pxl[i
    ],(40,40))
353                                         pxl[i]=resized_image
354
355                     else:
356                         if not(len(zs[str(i)]) in range(0,30)) and no_of_dots[
    i]<=3 and position(pxl[i]):
357                             RD = xyz_r(pxl[i])
358                             LD = xyz_l(pxl[i])
359
360                             h1, w1 = pxl[i].shape[:2]
361                             black_image = np.zeros((pxl[i].shape[0], 42-pxl[i
    ].shape[1],3))
362                             h2, w2 = black_image.shape[:2]
363
364                             #create empty matrix
365                             vis = np.zeros((max(h1, h2), w1+w2,3), np.uint8)
366                             if RD != None and LD != None:
367                                 if RD>LD:
368                             #combine 2 images
369                                     vis[:h1, :w1,:3] = pxl[i]
370                                     vis[:h2, w1:w1+w2,:3] = black_image
371                                     resized_image=cv2.resize(vis,(40,40))
372                                     pxl[i]=resized_image
373                                     rows,cols,w=pxl[i].shape
374                                     for x in range(rows):
375                                         for y in range(cols):
376                                             pixel=pxl[i][x][y]
377                                             if pixel[2]>pixel[1]
    and pixel[2]>pixel[0]:
378                                                 pxl[i][x][y
    ]=[0,0,0]
379
380                                 elif LD>RD:
381                                     vis[:h2, :w2,:3] = black_image
382                                     vis[:h1, w2:w1+w2,:3] = pxl[i]
383                                     resized_image=cv2.resize(vis,(40,40))
384                                     pxl[i]=resized_image
385                                     rows,cols,w=pxl[i].shape
386                                     for x in range(rows):
387                                         for y in range(cols):
388                                             pixel=pxl[i][x][y]
389                                             if pixel[2]>pixel[1]
```

```
    and pixel[2]>pixel[0]:
390                                                                 pxl[i][x][y
    ]=[0,0,0]
391
392                                 else:
393                                         resized_image=cv2.resize(pxl[i
    ],(40,40))
394                                         pxl[i]=resized_image
395
396             for i in range(len(pxl)):
397                 resized_image=cv2.resize(pxl[i],(40,50))
398                 pxl[i]=resized_image
399
400             w=[]
401             cvt=[]
402             dic={}
403             srt_dic={}
404             #appending the coordinates of all the white pixels in the image in
     dic[str(i)]
405             for i in range(len(pxl)):
406                 dic[str(i)]=[]
407                 srt_dic[str(i)]=[]
408
409             def contains_white(img,x):
410                 ret, threshold = cv2.threshold(img,200,255,cv2.THRESH_BINARY)
411                 h,w,l=img.shape
412                 for i in range(h):
413                     for j in range(w):
414                         a = threshold[i][j]
415                         if a.all()==255:
416                             dic[str(x)].append((j,i))
417                 return 0
418
419             for x in range(len(pxl)):
420                 result= contains_white(pxl[x],x)
421
422             for x in range(len(pxl)):
423                 dic[str(x)]=sorted(dic[str(x)], key=lambda tup: tup[1])
424
425             #end_pts contain the coordinates of 1st and last white pixel in
    the image
426             end_pts=[]
427             for i in range (len(pxl)):
428                 if (len(dic[str(i)]) != 0) :
429                     end_pts.append((dic[str(i)][0], (dic[str(i)][len(dic[str(i
    )])-1])))
430                 else:
431                     end_pts.append(())
432
433             for i in range(len(pxl)):
434                 cv2.line(pxl[i],((pxl[i].shape[1])//2,0),(((pxl[i].shape[1])
    //2),pxl[i].shape[0]),(0,0,255),2)
435                 cv2.line(pxl[i],(0,(pxl[i].shape[0])//3),(40,(pxl[i].shape[0])
    //3),(0,0,255),2)
```

```
436                 cv2.line(pxl[i],(0,(2*(pxl[i].shape[0])//3)),(40,(2*(pxl[i].
    shape[0])//3)),(0,0,255),2)
437                 #dividing each image into 6 equal rois
438                 w.append((pxl[i][0:(pxl[i].shape[0])//3,0:(pxl[i].shape[1])
    //2],      pxl[i][(pxl[i].shape[0])//3:2*((pxl[i].shape[0])//3),0:(pxl[i].
    shape[1])//2],          pxl[i][2*((pxl[i].shape[0])//3):pxl[i].shape[0],0:(
    pxl[i].shape[1])//2] ,      pxl[i][0:(pxl[i].shape[0])//3,(pxl[i].shape[1])
    //2:(pxl[i].shape[1])],      pxl[i][(pxl[i].shape[0])//3:2*((pxl[i].shape
    [0])//3),(pxl[i].shape[1])//2:(pxl[i].shape[1])] ,       pxl[i][2*(pxl[i].
    shape[0])//3:pxl[i].shape[0],(pxl[i].shape[1])//2:(pxl[i].shape[1])]))
439                 cvt.append([0,0,0,0,0,0])
440
441             for i in range(len(w)):
442                 for j in range(6):
443                     for x in range(w[i][j].shape[0]):
444                         for y in range(w[i][j].shape[1]):
445                             pixel=w[i][j][x][y]
446                             if pixel[0]>100 and pixel[1]>100 and pixel[2]>100:
447                                 q=1
448                                 cvt[i][j]=1
449                                 continue
450
451             for i in range(len(cvt)):
452                 a="".join(map(str,cvt[i]))
453                 cvt[i]=a
454             #defining the classifier set as follows
455             d={'numbers': '001111','capital': '000001', 'decimal': '000101'
    }
456             alpha={'a': '100000', 'c': '100100', 'b': '110000', 'e': '100010',
     'd': '100110', 'g': '110110', 'f': '110100', 'i': '010100', 'h': '110010'
    , 'k': '101000', 'j': '010110', 'm': '101100', 'l': '111000', 'o': '101010
    ', 'n': '101110', 'q': '111110', 'p': '111100', 's': '011100', 'r': '
    111010', 'u': '101001', 't': '011110', 'w': '010111', 'v': '111001', 'y':
    '101111', 'x': '101101', 'z': '101011'}
457             n={'0':'010110', '1':'100000', '2':'110000', '3':'100100','4':'
    100110' ,'5':'100010' ,'6':'110100' ,'7':'110110'  ,'8':'110010' ,'9': '
    010100'}
458             c={',':'010000','.':'010011','!':'011010','?':'011001',';':'011000
    ','}
459
460             #character recognition
461             for ch,valu in alpha.items():
462                 if cvt[0]==valu:
463                     f.write (ch)
464                     continue
465             for ch,valu in c.items():
466                 if cvt[0]==valu:
467                     f.write (ch)
468                     continue
469
470             for i in range(1,len(cvt)):
471                 try:
472                     if cvt[i-1]==d['numbers']:
473                         f.write (list(n.keys())[list(n.values()).index(cvt[i])
```

```
    ])
474                         continue
475
476                     elif cvt[i-1]==d['capital']:
477                         z=string.lowercase.index(alpha.keys()[alpha.values().
    index(cvt[i])])
478                         f.write (string.uppercase[z])
479                         continue
480
481                     else:
482                         for ch,valu in alpha.items():
483                             if cvt[i]==valu:
484                                 f.write (ch)
485                                 continue
486
487                         for ch,valu in c.items():
488                             if cvt[i]==valu:
489                                 f.write (ch)
490                                 continue
491
492                 except ValueError:
493                     pass
494
495             f.write (' ')
496
497         f.close()
498
499 ################################
500 # general global variables#
501 ################################
502 now = datetime.datetime.now()
503 flag=0
504 ##########################################
505 # project specific global variables#
506 ##########################################
507 state_input = 1
508 state_exit = 7
509 input_key = ' '
510 ########################################
511 #function to find which key pressed#
512 ########################################
513 def state_input_function():
514     global input_key
515     global state
516     global state_input
517     global state_realtime
518     global state_toggle
519     global state_scannedimage
520     global state_start
521     while(True):
522         os.system ('espeak "Press any key" ')
523         input_key = my_Input()
524         print("current key = ", input_key)
525
```

```
526         if "S" in input_key:
527             state = state_capture
528         elif "R" in input_key or "F" in input_key or "B" in input_key:
529             state = state_sonu
530         elif "T" in input_key:
531             state = state_toggle
532         elif "M" in input_key:
533             state = state_meaning
534         elif "E" in input_key:
535             state = state_exit
536         else:
537             continue
538 
539         print("from inputter state = ", state, ", user pressed : ", input_key)
540         break
541 
542 ##############################
543 #function for user manual#
544 ##############################
545 def user_manual():
546   t= now.strftime('%H')
547   if t< '12':
548             os.system ('espeak -s 150 "Good Morning! I am eReader, your
      reading assistance. Lets read something today " ')
549 
550   elif  t >='12' and t<='14':
551       os.system ('espeak -s 150 "Good Afternoon! I am eReader, your reading
      assistance. Lets read something today " ')
552 
553   else:
554       os.system ('espeak -s 150 "Good Evening! I am eReader, your reading
      assistance. Lets read something today " ')
555 
556   os.system ('espeak -s 150 "Welcome to the User instruction guide! To move
      forward to the next image,press F" ')
557   os.system ('espeak -s 150 "To move backward to the previous image,press B" '
      )
558   os.system ('espeak -s 150 "To repeat the same image,press R " ')
559 
560 
561 def mainloop():
562     global state
563     global toggleMode
564     print("S - Pre scanned")
565     print("F - Forward")
566     print("B - Backward")
567     print("R - Repeat")
568     print("M - Real Time")
569     os.system ('espeak -s 150 "press R to read the user manual! If you want to
      skip the instruction manual press any other key!" ')
570     inputKey = my_Input()
571     if(inputKey =='R'):
572         user_manual()
573     elif(inputKey == 'E'):
```

```
574         print("Exiting")
575         os.system ('espeak -s 150 "Preparing to shutdown" ')
576     else:
577        os.system ('espeak -s 150 "user manual skipped" ')
578     os.system ('espeak -s 150 "Press S to hear previously stored images" ')
579     os.system ('espeak -s 150 "Press M to hear realtime scanned images" ')
580     inputKey = my_Input()
581     if(inputKey =='S'):
582             flag1 =0
583             img_counter=0
584     elif(inputKey =='M'):
585             flag1 =1
586             r_counter=0
587
588     while flag1==0:
589             img= cv2.imread("/home/pi/Desktop/INPUT/{}.png".format(img_counter
        ),0)
590             rows = processImage(img)
591             f=open("/home/pi/Desktop/eReader/{}.txt".format(img_counter),"w+")
592             VertSeg(f,rows)
593             with open("/home/pi/Desktop/eReader/{}.txt".format(img_counter), '
        r') as myfile:
594                     data = myfile.read()
595             tts = gTTS(data, lang='en')
596             tts.save("/home/pi/Desktop/eReader/{}.mp3".format(img_counter))
597             if (img_counter==0):
598                     os.system ('espeak -s 150 -f /home/pi/Desktop/eReader/{}.
        txt'.format(img_counter))
599                     img_counter += 1
600             elif (img_counter!=0):
601                     os.system ('espeak -s 150 -f /home/pi/Desktop/eReader/{}.
        txt'.format(img_counter))
602                     os.system ('espeak -s 150 "Press B to hear previous image"
        ')
603                     os.system ('espeak -s 150 "Press F to hear next image"')
604                     os.system ('espeak -s 150 "Press R to hear same image"')
605                     inputKey = my_Input()
606                     if(inputKey =='B'):
607                             os.system ('espeak -s 150 -f /home/pi/Desktop/
        eReader/{}.txt'.format(img_counter-1))
608
609                     elif(inputKey =='R'):
610                             os.system ('espeak -s 150 -f /home/pi/Desktop/
        eReader/{}.txt'.format(img_counter))
611                             img_counter += 1
612                     elif(inputKey =='F'):
613                             img_counter += 1
614
615             if (img_counter>4):
616                     os.system ('espeak -s 150 "No more images to process"')
617                     break
618     while flag1 ==1:
619             img= cv2.imread("/home/pi/Desktop/REALINPUT/{}.jpg".format(
        r_counter),0)
```

```
620             file = realtime(img,r_counter)
621             with open("/home/pi/Desktop/REALINPUT/{}.txt".format(r_counter), '
    r') as myfile:
622                     data = myfile.read()
623             tts = gTTS(data, lang='en')
624             tts.save("/home/pi/Desktop/REALINPUT/{}.mp3".format(r_counter))
625             if (r_counter==0):
626                     os.system ('espeak -s 150 -f /home/pi/Desktop/REALINPUT
    /{}.txt'.format(r_counter))
627                     r_counter += 1
628             elif (r_counter!=0):
629                     os.system ('espeak -s 150 -f /home/pi/Desktop/REALINPUT
    /{}.txt'.format(r_counter))
630                     os.system ('espeak -s 150 "Press B to hear previous image"
    ')
631                     os.system ('espeak -s 150 "Press F to hear next image"')
632                     os.system ('espeak -s 150 "Press R to hear same image"')
633                     inputKey = my_Input()
634                     if(inputKey =='B'):
635                             os.system ('espeak -s 150 -f /home/pi/Desktop/
    REALINPUT/{}.txt'.format(r_counter-1))
636
637                     elif(inputKey =='R'):
638                             os.system ('espeak -s 150 -f /home/pi/Desktop/
    REALINPUT/{}.txt'.format(r_counter))
639                             r_counter += 1
640                     elif(inputKey =='F'):
641                             r_counter += 1
642
643             if (r_counter>2):
644                     os.system ('espeak -s 150 "No more images to process"')
645                     break
646
647 state = state_input
648 my_setupInput()
649 mainloop()
650 my_safeShutdown()
```