VOICE - ENABLED TEXT READER FOR VISUALLY IMPAIRED

A PROJECT REPORT

Submitted by

BALAJI. S. D

BHARANI DHARAN. K

DIVAKARAN. J

GURU VENKATESH. S

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

SARANATHAN COLLEGE OF ENGINEERING

(An Autonomous Institution)

Tiruchirappalli 620 012

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2025

SARANATHAN COLLEGE OF ENGINEERING

(An Autonomous Institution)

Tiruchirappalli 620 012

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**VOICE - ENABLED TEXT READER FOR VISUALLY IMPAIRED**" is the bonafide work of "**BALAJI. S. D, BHARANI DHARAN. K, DIVAKARAN. J, GURU VENKATESH. S**" who carried out the project work under my supervision.

SIGNATURE

Dr. M. SANTHI

HEAD OF THE DEPARTMENT

Professor

Department of Electronics & Communication Engineering

Saranathan College of Engineering,

Venkateswara Nagar, Trichy,

Tamil Nadu - 620012

SIGNATURE

Dr. V. MOHAN

SUPERVISOR

Professor

Department of Electronics & Communication Engineering

Saranathan College of Engineering,

Venkateswara Nagar, Trichy,

Tamil Nadu - 620012

Submitted for the project viva-voce examination held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

CERTIFICATE OF EVALUATION

College Code : 8138

College Name : Saranathan College of Engineering

Branch : Electronics and Communication Engineering

Semester : VIII

Subject : EC3811 – Project Work

S. NO	Name of the Students	Title of Project	Name of the Supervisor with Designation
1.	Balaji S D (813821106013)	VOICE - ENABLED TEXT READER FOR VISUALLY IMPAIRED	Dr. V. Mohan M.E, Ph.D Professor, Department of ECE
2.	Bharani Dharan K (813821106015)		
3.	Divakaran J (813821106023)		
4.	Guru Venkatesh S (813821106036)		

The report of the project work submitted by the above students in partial fulfilment for the award of Degree of Bachelor of Engineering in Electronics and Communication Engineering was confirmed to be the work done by the above students and evaluated on _____.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We would like to express our deep sense of gratitude to our Secretary **Shri. S. RAVINDRAN**, Saranathan College of Engineering for his sincere endeavor in educating us in our esteemed institution.

We thank our Principal **Dr. D. VALAVAN**, Saranathan College of Engineering for giving permission to do this project.

We sincerely thank our Head of the Department (R&D) **Dr. R. NATARAJAN**, Saranathan College of Engineering, for his constructive information and valuable suggestions during the course of project.

We are extremely thankful to our Head of the Department, **Dr. M. SANTHI**, Department of Electronics and Communication Engineering, for her support and inspiration, and for the constant encouragement throughout the progress of the work.

We are immensely pleased to thank our supervisor **Dr. V. MOHAN**, Professor, for gracefully accepting us as project students. The whole period of our project has been an excellent learning curve by virtue of his wisdom, advice, support, motivation, and valuable suggestions.

We are expressing our gratefulness to our beloved Project Coordinator **Dr. P. SHANMUGAPRIYA**, Professor and **Ms. M. ANTHUVAN LYDIA**, Assistant Professor for coordinating the activities and encouragement during the project work.

A special thanks to our panel members for their valuable suggestions and interest at every stage of the project.

ABSTRACT

Visually challenged people have difficulty accessing textual material, particularly in areas where costly, cloud-based optical character recognition (OCR) systems are impractical. Current solutions, such as Microsoft Seeing AI and Google Lens, rely on cloud processing, which necessitates an internet connection and frequently causes delays. Real-world usability is further limited by these systems' difficulties with handwritten content, complex backgrounds, low-quality ink, and low-resource languages. A low-cost, offline, multilingual text reader that uses a Raspberry Pi 5© to solve these problems. Our technique uses median blurring to reduce noise and Contrast Limited Adaptive Histogram Equalization (CLAHE)© to boost contrast in text recognition. It effectively recognizes English, Kannada, and Telugu using EasyOCR©, which improves adaptability to handwritten text, mixed scripts, and fonts. To improve readability and coherence, we incorporate IndicNLP© for text normalization and SymSpell for spell correction. Text-to-Speech (TTS) output is enhanced by a rule-based transliteration model for Indic characters, which provides fluid and natural voice synthesis for unambiguous audio feedback. Our system outperforms cloud-dependent alternatives with its high recognition accuracy, real-time processing, and improved speech quality. Providing a self-sustaining, inclusive reading experience, it is completely offline, affordable, and accessible, enabling visually impaired individuals to independently access both printed and digital literature.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

SYMBOL	DESCRIPTION
I_{gray}	Grayscale value
R	Red channel
G	Green channel
B	Blue channel
N	Total number of pixels in one tile
h_i	Histogram bin count for intensity level
$G(x, y)$	Value of the Gaussian kernel at position (x,y)
(x, y)	Pixel offsets from the center of the kernel (e.g., for 3×3: x,y ∈ {−1,0,1})
σ	Standard deviation controls the amount of blur (larger σ = more blur)
π	Mathematical constant ≈ 3.14159
e	Euler's number (base of natural logarithms) ≈ 2.71828

LIST OF ABBREVIATIONS

ABBREVATION	DESCRIPTION
CDF	Cumulative Distributive Function
CER	Character Error Rate
CLAHE	Contrast Limited Adaptive Histogram Equalization
CNN	Convolutional Neural Network
CRAFT	Character Region Awareness for Text Detection
CRNN	Convolutional Recurrent Neural Networks
CTC	Connectionist Temporal Classification
DB	Differentiable Binarization
FPN	Feature Pyramid Network
LSTM	Long Short-Term Memory
OCR	Optical Character Recognition
OS	Operating System
ResNet	Residual Network
RNN	Recurrent Neural Network
TTS	Text-to-Speech
WER	Word Error Rate

CHAPTER 1
INTRODUCTION

One of the greatest challenges encountered by individuals with visual impairments is considered to be access to written material, through which their participation in daily life, employment, and education is affected [1]. Specifically, solutions with limitations, including cost and lack of dynamic content support, are offered by traditional approaches such as audiobooks and Braille [3]. Independence is also often restricted for individuals with visual impairments due to the accessibility and availability of these types of solutions. In a number of contexts, OCR technology and text-to-speech have been examined as accessible alternative technologies for addressing this problem [2].

1.1 TEXT TO SPEECH CONVERSION - AN OVERVIEW

The application of Text-to-Speech (TTS) has increasingly been seen as critical in closing the accessibility gap by virtue of advances in assistive technologies. Optical Character Recognition (OCR) solutions like Tesseract OCR© are challenged by extracting text from handwritten materials, multilingual text, or in real-time on low-powered embedded platforms [7]. While greater accessibility can be facilitated by an OCR-based aid, as previous studies suggest, the quality of the extracted text may be impacted negatively by low-quality images, font variations, and complex layouts [5]. The identified text is then converted into speech outputs by TTS technology, making printed or digital content available for people with visual impairments [17]. However, several challenges are still faced by current OCR technologies like Tesseract, particularly for real-time processing of handwritten or multilingual texts and during implementation on low-powered embedded devices [11]. High accuracy issues are also faced by Tesseract-based solutions when unprecedented fonts and complicated layouts are being processed [12].

Presently, widely adopted text reading assistive technologies, such as Google Lens, Microsoft Seeing AI and Envision AI, are based on OCR and cloud services, resulting in reliance on the internet and the introduction of latency [10]. Difficulties are also often encountered with low-resource languages, and consistent performance is not always achieved when the text is presented on a complex background, printed with less-than-ideal quality, or written in a handwritten format [8].

1.2 PROBLEM DESCRIPTION

Despite being widely used for text extraction, significant accuracy issues are faced by traditional OCR systems like Tesseract OCR when handwritten documents, non-standard fonts, and complex page layouts are processed [2]. Moreover, cloud-based Text-to-Speech (TTS) services, such as Google TTS and Amazon Polly, are considered unsuitable for offline use due to privacy concerns and the requirement of a constant internet connection [17]. While cloud resources are relied upon by most implementation thus restricting accessibility in offline scenarios real-time OCR-based assistive tools have recently been introduced by researchers to enable printed text to be read aloud to visually impaired individuals [16].

1.3 PROBLEM FORMULATION

The main difficulty is developing a multilingual reader to read your text that is also fully offline. In addition to needing a high degree of accuracy and natural voice without processing in the cloud, the reader needs to vocalize on the fly whether it is reading printed physical text or tense handwritten text and also from multiple scripts as the printed or handwritten text can be of varied quality and hold other quality standards.

1.4 SUMMARY OF THE PROPOSED APPROACH

Our system has a voice-activated and totally offline text-to-speech framework for a Raspberry Pi 5© as a solution to meet these challenges. Some of the key features are:

a) Multilingual OCR Processing: EasyOCR© [4][13] for high-accuracy text recognition in English, Kannada, and Telugu. Prior studies have demonstrated that deep learning-based OCR models outperform traditional OCR engines in recognizing non-standard scripts and handwritten text.
b) Enhanced Image Preprocessing: Contrast Limited Adaptive Histogram Equalization (CLAHE)© for contrast enhancement and median blurring for noise reduction, improving OCR accuracy in poor lighting conditions. Previous research highlights that preprocessing techniques significantly enhance the OCR accuracy of low-resolution and degraded images.
c) Text Refinement: SymSpell for fast spell correction and IndicNLP for text normalization to correct inconsistencies in extracted text. Prior work has shown that integrating linguistic models with OCR can improve the readability and accuracy of extracted text, especially for noisy or distorted characters.
d) Rule-Based Transliteration: A Multilingual Attentive Transliteration System [16], enabling accurate pronunciation of Indic scripts, enhancing speech clarity. Transliteration models have been proven to enhance the accessibility of multilingual content for visually impaired individuals by providing phonetic clarity.
e) Offline Speech Synthesis: Pyttsx3© [11] is implemented for text-to-speech (TTS) conversion to provide natural-sounding voice output without requiring an internet connection. Research indicates that offline speech synthesis engines like pyttsx3 are well-suited for low-powered embedded systems, as they offer reliable performance while maintaining speech intelligibility.

CHAPTER 2

LITERATURE SURVEY

2.1 READING ASSISTANT FOR THE VISUALLY IMPAIRED

This paper describes an approach to image recognition and speech synthesis using a small computer specifically Raspberry Pi© that allows individuals with visual disabilities to read printed text. The system takes pictures using an embedded camera, passes the images to Tesseract OCR to process, and then rest of the data is converted to speech output by a text-to-speech (TTS) engine [3]. The approach described in the paper utilizes Raspbian OS© running Python and is capable of working with Raspberry Pi© hardware. Their approach primarily shows a reliance on rule-based OCR with limited capabilities to convert handwritten text, font styles, or low light conditions, and it did not show progress towards multilingual. In contrast, my approach uses EasyOCR© with CRNN (Convolutional Recurrent Neural Network)© to improve recognition for handwritten, stylized text and multiple (up to 70) languages.

An approach has been developed in which frequently seen noise patterns are removed from images of printed text, incorporating adaptive thresholding, noise and blur residual average reduction, and CLAHE© contrast enhancement to improve OCR performance when images are captured in low light, are of low quality, or contain degraded text. Furthermore, SymSpell© has been incorporated into the approach, with OCR text being spell-checked before TTS conversion is integrated, thereby providing a better reading experience. Overall, enhancements have been made over the referenced paper's approach, particularly in terms of accuracy and in the consideration of technologies implemented separately from the TTS engine.

2.2 MACHINE TRANSLITERATION - A REVIEW OF LITERATURE

Machine transliteration, is an essential part of cross-language text processing since it involves mapping words in one script to another script while retaining phonetic characteristics. The paper separates transliteration approaches into grapheme-based, phoneme-based, and hybrid approaches [6]. It also describes common methods of operationalization, including using rule-based systems, SMT, FST, HMM, CRF, SVM, and neural networks. Unfortunately, the paper lacked experimental validation, benchmark datasets, and performance comparisons, which makes it difficult to gauge the effectiveness of transverse methods and recommend a few approaches over others. The authors do not discuss modern transformer-based transliteration models at all, which have improved transliteration accuracy significantly. Key challenges presented are different scripts (LTR vs. RTL), phonological mismatches, multiple valid retrials, and little or no available resources for under-represented languages. In addition, standard benchmarks do not exist for transliteration models which has resulted in off-system evaluation methods. Finally, most systems were unable to simply incorporate context of transliteration, which is particularly critical when resolving named entities and disambiguating and/or ambiguous terms.

To mitigate the challenges outlined, an effective approach tailored to low-resource settings has been provided through the implementation of a custom transliteration code combined with the Indic transliteration module and pyttsx3 for speech synthesis. In addition, it is believed that transliteration consistency across Tamil, Telugu, and Kannada can be further improved through phoneme-aware normalization. Furthermore, transliteration accuracy can be enhanced through the exploration of transfer learning methods derived from high-resource language models applied to low-resource models. Due to the limited hardware capacity of the Raspberry Pi 5©, the transliteration system will need to be

optimized using either an efficient rule-based or hybrid type of system to ensure real-time computation within computational constraints. A custom-built transliteration benchmark dataset for the OCR system is expected to be developed, enabling the transliteration accuracy to be fine-tuned and evaluated. Finally, if required, lightweight transliteration models optimized for Raspberry Pi© can be employed to contribute to performance without sacrificing efficiency.

2.3 IMAGE TEXT TO SPEECH CONVERSION WITH RASPBERRY-PI USING OCR

In this article, an assistive system based on a Raspberry Pi© is introduced to provide support for visually impaired individuals by converting text from images into speech. The process is initiated by capturing an image using the 5 MP Raspberry Pi Camera Module© attached to the Camera Serial Interface (CSI) port. Following image capture, preprocessing is conducted using OpenCV©, where a Gaussian blur filter is applied for noise removal, and the Laplacian operator is used to assess edge sharpness and blurriness for clearer image acquisition. After preprocessing, text is extracted from the image using Pytesseract OCR (Optical Character Recognition) © [11], producing machine-readable text. Once the text is generated, speech output is produced by eSpeak, a small and efficient TTS synthesizer, which can then be played through speakers or earphones. The hardware used in the project consisted of a Raspberry Pi 3B© (equipped with a quad-core 1.4 GHz processor) running Raspberry Pi OS©, and implementation was carried out using Python.

Although the system has been found to function successfully, several limitations have been identified. Handwritten text, non-standard fonts, and low-light conditions are not easily processed by Pytesseract OCR©, which negatively impacts real-world accuracy. Additionally, support for multiple languages has not been included, making the system unsuitable for non-English speakers. Image

preprocessing has relied on Gaussian blur and Laplacian variance, and although these methods have contributed to improved visibility in degraded images, their impact has been somewhat limited. The speech generated by eSpeak has been reported to sound robotic and unnatural, which detracts from the user's engagement with the auditory experience. Moreover, delays have been introduced due to the limited processing power of the Raspberry Pi 3B©, negatively affecting real-time performance.

To address these limitations, Pytesseract© has been replaced by EasyOCR©, which utilizes a CRNN (Convolutional Recurrent Neural Network)© for improved text recognition accuracy, particularly with handwritten text and multilingual content. In contrast to previous work that applied basic filtering, image preprocessing was enhanced through the use of advanced techniques such as CLAHE© (Contrast Limited Adaptive Histogram Equalization) for contrast enhancement, adaptive thresholding for binarization, and targeted noise reduction for noisy images. Through these combined enhancements, the system has been made more accurate, responsive, and accessible for blind and visually impaired users.

2.4 MATRA - A MULTILINGUAL ATTENTIVE TRANSLITERATION SYSTEM FOR INDIAN SCRIPTS

MATRA is a multilingual transliteration system designed for Indian scripts using a modified transformer model. It improves transliteration between English, Hindi, Bengali, Kannada, and Tamil, achieving 80.7% top-1 accuracy and 93.5% phonetic accuracy [16]. The system is trained on NEWS 2018 and NEWS 2012 datasets (113k word pairs) and employs bi-directional training for English-Indic and Indic-English transliteration.

It evaluates performance using phonetic accuracy, character error rate (CER), and BLEU scores. However, MATRA's deep learning-based approach

requires high computational power, and its multi-stage inference process increases processing time.

Our method provides lightweight, offline-compatible transliteration that's well-suited for the Raspberry Pi 5©. Unlike MATRA's approach that utilized deep learning models to create a text sequence, our systems make use of simple custom transliteration code, the Indic transliteration module, and pyttsx3 for TTS to provide transliteration that could be done in real-time with almost negligible latency. By utilizing rule-based and statistical approaches, our technology does not rely on larger datasets for transliteration, allowing for customization and adaptability. Our system facilitates transliteration directly between Tamil, Telugu, and Kannada text so it will be more readable and usable. Unlike MATRA's approach that used multi-stage inference as a main component of the transliteration, low latency transliteration while adding a multi-language OCR capability to extract both the text and pronunciation without any interruptions.

2.5 IMAGE TEXT DETECTION AND DOCUMENTATION USING OCR

This article investigates the implementation of Optical Character Recognition (OCR) technology for use in document management and data entry. The authors implemented Pytesseract, an open-source OCR engine, to extract text data from images and output it to a CSV file using a CSV formatted library in Python [1]. The article documents how OCR can help streamline and process data, minimize manual labour through automation, and reduce errors to improve accuracy in the management of files. The outcome highlights OCR is becoming increasingly more useful for automating data entry tasks and shows a pathway to advancements in the potential for OCR technology [1]. Despite the successful advantages of the Pytesseract based workflow it has shortcomings with low accuracy in incorporating handwritten text data, noisy images, not handling

complex layout files or multilingual documents, lower-resolution images, and requires complete preprocessing to provide reliable extraction results. To address these limitations, Pytesseract was replaced with EasyOCR© a CRNN© (Convolutional Recurrent Neural Network) architecture to provide sufficient text recognition coverage especially with handwritten, stylized, and multilingual texts. Additionally, I implemented advanced image processing techniques, including CLAHE© (Contrast Limited Adaptive Histogram Equalization) related to contrast enhancement, adaptive thresholding, and noise to develop better accuracy in challenging environments related to lower acids resource and, as a result, being able to adapt to being able to have successful outcomes. Additionally, post-processing techniques like SymSpell© for spell correction model for contextual refinement further enhance text recognition accuracy, making my approach more robust and reliable for automated document management.

2.6 MULTILINGUAL TEXT RECOGNITION SYSTEM

This paper explores the use of OCR technology [9] to assist visually impaired individuals by converting printed and electronic text into speech output. The study specifically evaluates two OCR engines namely Tesseract OCR and EasyOCR© by comparing their accuracy in recognizing text from business cards in English, Bengali, and Tamil. The system extracts all legible data using both tools, followed by a comparative analysis based on accuracy parameters to determine which OCR engine performs better for multilingual text recognition.

While EasyOCR© outperforms Tesseract in recognizing complex scripts and handwritten text, its accuracy is still affected by low-resolution images, noisy backgrounds, and character misinterpretations in Bengali and Tamil scripts. My approach improves accuracy with advanced preprocessing (CLAHE©, adaptive thresholding), refines output using SymSpell for spell correction.

2.7 DESIGN AND DEVELOPMENT OF WEARABLE TEXT READING DEVICE FOR VISUALLY IMPAIRED PEOPLE

The project represents a wearable text-reading device for the visually impaired that performs text recognition and converts it to speech using a camera-based Raspberry Pi© system [10]. This system recognizes text using Optical Character Recognition (OCR) algorithms and synthesizes speech for audio output. In addition, a specialized dataset of 62,992 images was created to develop the OCR model, where each image was 128x128 pixels and contained characters, numbers, and symbols in four distinct font styles. The project uses preprocessing on images to maximize the effectiveness of OCR. Tesseract served as the OCR engine for this video and was trained using various pre-trained models to help with the recognizing process. The training used TensorFlow machine learning framework and it utilized deep learning & image prepossessing techniques [10]. The Tesseract OCR engine was tested on a variety of inputs including invoices, digital text, and handwritten characters, achieving an accuracy of 94.93%, which demonstrates the system's effectiveness in real-world scenarios. While achieving respectable accuracy, this approach has shortcomings. Tesseract OCR, while very effective for printed text, encounters trouble when attempting to interpret handwritten text. If the background of the text is complex tesseract will also struggle to achieve good accuracy, which results in lower effectiveness for OCR in real-world applications of unstructured texts/documents. Given the resolution of the dataset, a 128x128pixels image, the resolution of the text in the picture could greatly impact the ability to recognize fine details, especially smaller text, or text that was distorted. Additionally, pre-trained models were used but not fine-tuned for real-time performance, leading to delays in processing on Raspberry Pi© hardware. Furthermore, speech synthesis was not emphasized, potentially resulting in robotic or unnatural audio output.

2.8 AN END-TO-END TRAINABLE NEURAL NETWORK FOR IMAGE-BASED SEQUENCE RECOGNITION AND ITS APPLICATION TO SCENE TEXT RECOGNITION

Convolutional Neural Networks (CNNs) are fundamental to Optical Character Recognition (OCR) based on deep learning as they automatically extract spatial features from images [4]. The EasyOCR© approach takes shape as two-compartment process of text detection, followed by text recognition. An initial phase of text detection used the CRAFT model, which outputs character regions and then groups those character regions together as words. Then those detected character regions are processed through a ResNet LSTM CTC based recognition model [4] ResNet extracts visual features, LSTM captures sequential dependencies, and CTC encodes the text, all without the need of precise segmentation of the character. This multi-model approach ensures accurate recognition of multilingual text, including handwritten text.

In terms of text detection, CNN-based models like CRAFT© produce character region maps and affinity score maps for precise localization and grouping of text. Differentiable Binarization (DB), using adaptive thresholding for better text segmentation, is then employed for added accuracy. The DB network has a ResNet-based backbone and a Feature Pyramid Network (FPN) to accommodate multi-scale text. This allows the text to be reliably extracted from noisy, distorted images. Finally, the optimized binarized output is sent to a CRNN-based recognition model.

EasyOCR© employs a CRNN to accomplish text recognition that combines CNN spatial feature extraction capabilities and RNN's ability to model sequences. In contrast to a standard CNN, CRNN captures spatial dependencies by converting visual features into sequential representations arising from recognizing text as sequences of visual symbols. The recurrent component of the

model is composed of a stack of LSTM layers that track long-range dependencies while providing context and allowing the model to infer missing or ambiguous characters in text. Such modelling capabilities are important for recognizing complex text and handwriting.

The final decoding stage employs Connectionist Temporal Classification (CTC), which maps variable-length input sequences to structured text outputs. CTC resolves alignment issues by allowing flexible character placement and handling inconsistent spacing or distortions in images. It introduces a blank token to merge duplicate predictions, improving accuracy. This end-to-end approach enables OCR systems like EasyOCR to process multilingual text efficiently without requiring detailed character-level annotations, making it effective for real-world applications.

CHAPTER 3

MATHEMATICAL FOUNDATION OF THE PROPOSED SYSTEM

3.1 GRAYSCALE CONVERSION

The Grayscale conversion is a crucial pre-processing step in our OCR-based assistive system for visually impaired users. The image captured is then converted into grayscale it simplifies image data by reducing computational complexity while preserving essential text features.

In the grayscale conversion process, the RGB components of the captured image are used to produce a single grayscale intensity value. The red (R), green (G), and blue (B) channels are combined using a weighted sum. Equation 3.1 is used for this conversion

$$I_{gray} = 0.299R + 0.587G + 0.114B \quad \rightarrow (3.1)$$

Where

- The red channel (R) is assigned a weight of 0.299.
- The green channel (G) is assigned a weight of 0.587.
- The blue channel (B) is assigned a weight of 0.114.

This weighted sum reflects the human eye's sensitivity to different colors, with the green channel being the most sensitive, followed by red, and blue being the least sensitive. As a result, the grayscale value (I_{gray}) is calculated by combining the RGB components with these weights to produce an intensity value that represents the pixel's brightness in grayscale.

In an RGB image, each pixel is made up of three-color channels: Red, Green, and Blue. Each channel stores an intensity value ranging from 0 to 255. These

intensity values are typically obtained during the process of image capture (using a camera or image sensor).

For example, an image could have the following RGB values

- Red (R) = 120
- Green (G) = 200
- Blue (B) = 150

These values are then used in the grayscale conversion formula (equation 1) to calculate the grayscale intensity for the image with R = 150, G = 100, and B = 200, the grayscale intensity would be

$$I_{gray} = 0.299(150) + 0.587(100) + 0.114(200)$$
$$I_{gray} = 44.85 + 58.7 + 22.8 = 126.35 \rightarrow (3.2)$$

Thus, the grayscale value for that image would be approximately 126(equation 3.2).

Impact on our Project

Since our project runs on a Raspberry Pi© with EasyOCR©, grayscale conversion helps optimize performance, reduce processing time, and enhance text recognition accuracy. It plays a critical role in ensuring that visually impaired users receive fast and accurate text-to-speech output, even in challenging real-world scenarios.

3.2 CLAHE-BASED CONTRAST ENHANCEMENT

The visibility of features, particularly in low-light or poor-quality images, is improved through contrast enhancement. In the OCR-based assistive system, CLAHE© (Contrast Limited Adaptive Histogram Equalization) is used to make the text clearer for improved recognition. After the image is converted into

grayscale, its contrast is enhanced using the CLAHE© technique. The steps involved in CLAHE is given below

Step 1: Image Tiling

The image size is specified as 640 pixels in width and 480 pixels in height. A tile grid size of (8, 8) is defined. Tile Size Calculation is as follows

$$\text{Tile Width} = \frac{640}{8} = 80 \text{ pixels} \qquad \text{Tile Height} = \frac{480}{8} = 60 \text{ pixels} \rightarrow (3.3)$$

The image is divided into 64 equal-sized tiles, each measuring 80 × 60 pixels. Each tile is independently processed. Through this approach, contrast enhancement is localized, enabling adaptation to variations in lighting and background across different regions of the image. This method is particularly effective in real-world text scenarios where uneven illumination is present.

Step 2: Local Histogram Equalization (Per Tile)

For each tile, a gray-level histogram is computed to represent the distribution of pixel intensities. The histogram is clipped at a predefined threshold, specified by the clipLimit, to avoid enhancement of noise. Any excess pixel counts beyond this threshold are redistributed uniformly across all histogram bins.

A Cumulative Distribution Function (CDF) is then calculated from the modified histogram. All pixels within the tile are subsequently remapped using the CDF. This enables the enhancement of local brightness contrast, making subtle text features more prominent. As a result, the contrast between text and background is improved, thereby facilitating more accurate character recognition in OCR systems.

Step 3: Mathematical Equation

Let h_i represent the histogram bin count for intensity level i, and N=80×60 be the total number of pixels in one tile. To prevent over-enhancement of noise, a clip limit of 1.5 is used. The histogram is then clipped based on a threshold, calculated using the equation 3.4

$$Threshold = \frac{clipLimit \times N}{256} = \frac{1.5 \times 4800}{256} \approx 28.13 \rightarrow (3.4)$$

Step 4: Compute CDF and Remap Intensities

The normalized CDF is computed using equation 3.5

$$CDF(i) = \sum_{j=0}^{i} \frac{h_j}{N} N \rightarrow (3.5)$$

New intensity values for pixels are then assigned based on this CDF

$$I_{new} = \lfloor CDF(i) \times 255 \rfloor \rightarrow (3.6)$$

Through this process, the contrast within each tile is enhanced, and text edges are highlighted, thereby improving clarity and separability of characters.

Step 5: Bilinear Interpolation Between Tiles

All 64 tiles are processed individually, transitions between neighbouring tiles are smoothed through bilinear interpolation. This blending technique is applied to ensure that no harsh boundaries are visible between adjacent regions. As a result, a final image is produced in which local contrast is enhanced uniformly, while a smooth and natural appearance is maintained. This makes the output image highly suitable for subsequent OCR processing.

Stage	Details
Image Size	640 × 480
tileGridSize	(8, 8)
Tile Size	80 × 60 pixels
Total Tiles	64
clipLimit	1.5
Clipping Threshold	Approximately 28 pixels per histogram bin

Table 3.1 Image Tiling and CLAHE Parameters Considered During Preprocessing

Table 3.1 describes the parameters that were used during preprocessing, where the image was divided into tiles and CLAHE© was applied. The contrast was enhanced and normalization was achieved by adjusting tile size, grid layout, and clipping threshold.

CLAHE© is a crucial pre-processing step in our OCR engine as it improves text readability in images taken with the camera. In addition to improving overall input image quality, CLAHE© has a large impact on the text recognition accuracy. The advantage of CLAHE is its ability to mitigate lighting differences in the environment across different environments, providing more adaptability and robustness across differing conditions. CLAHE© also works to decrease background noise so that text appears more distinct and clear to the OCR engine.

3.3 GAUSSIAN BLUR

Gaussian Blur remove noise from the image by applying various filters. It smooths an image by averaging pixel values with their neighbour's using a weighted Gaussian function. This helps reduce random noise, such as small grainy artifacts. Once the contrast has been enhanced, a Gaussian Blur is applied to the image to reduce noise and smoothen variations. This will assist in reducing

incorrect spatial references stemming from unwanted details that may interfere with OCR processing. This will improve the clarity of text while ensuring that the elements of a character are maintained to allow for more accurate character recognition. These preprocessing steps ensure that even text in challenging conditions, such as faded prints, shadows, or uneven lighting is correctly identified. The equation 3.7 used to compute each kernel value is given below and its variables are explained in the table 3.2 below,

$$G(x,y) = \frac{1}{2\pi\sigma^2} \times e^{-\frac{x^2+y^2}{2\sigma^2}} \rightarrow (3.7)$$

Symbol / Expression	Meaning
$G(x,y)$	Value of the Gaussian kernel at position (x,y)
(x,y)	Pixel offsets from the center of the kernel (e.g., for 3×3: x,y ∈ {−1,0,1})
σ	Standard deviation controls the amount of blur (larger σ = more blur)
π	Mathematical constant ≈ 3.14159
e	Euler's number (base of natural logarithms) ≈ 2.71828
$\frac{1}{2\pi\sigma^2}$	Normalization factor ensures the sum of all kernel values = 1
$-\frac{x^2+y^2}{2\sigma^2}$	Determines how far the pixel is from the center

Table 3.2 Explanation of Variables used in Gaussian Blur Equation

3.4 OCR (EASYOCR)©

In our project, we use EasyOCR© [4][13], the pre-processed image is then fed into the EasyOCR© engine, which analyses and extracts textual content.

EasyOCR identifies individual characters, combines them into words, and recognizes structured text formats. Unlike traditional OCR systems, EasyOCR© uses deep learning techniques that allow it to recognize handwritten as well as printed text with high accuracy. This block ensures that the extracted text is ready for language detection and conversion.

Optical Character Recognition (OCR) is a crucial component of our project, enabling the system to extract text from images and convert it into speech for visually impaired users. EasyOCR, a deep-learning-based OCR framework, is employed to detect and recognize text in multiple languages with high accuracy. It ensures real-time text recognition from a live webcam feed, allowing users to access printed or handwritten text seamlessly. The effectiveness of EasyOCR© is further enhanced by Convolutional Neural Networks (CNNs), which play a key role in image processing and feature extraction. CNNs help distinguish text from complex backgrounds, handle noise, and improve the readability of detected characters, ensuring that the OCR system performs efficiently in various conditions.

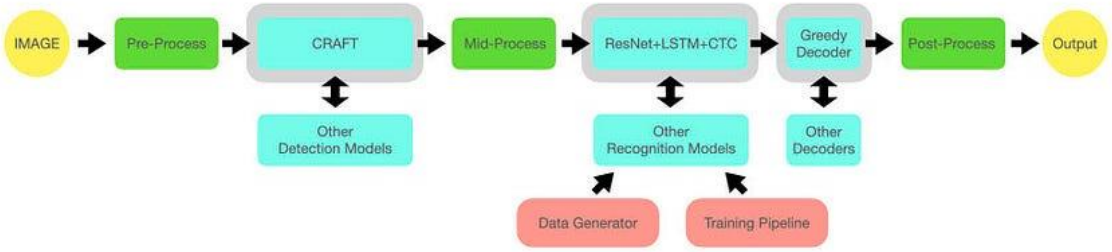


Figure 3.1 EasyOCR Framework

To accurately detect and recognize text from complex scenes, our system employs a powerful two-stage architecture combining CRAFT© for text detection and CRNN for text recognition as shown in the Figure 3.1 EasyOCR© integrates CRAFT and CRNN. CRAFT (Character Region Awareness for Text detection) identifies the precise locations of text within an image, regardless of font, size, or orientation. Once the text regions are detected, they are passed to

the CRNN (Convolutional Recurrent Neural Network), which extracts and decodes the character sequences using a combination of CNNs, RNNs, and CTC loss. This integrated approach ensures high accuracy in recognizing both printed and handwritten text in real-world conditions.

3.4.1 Detect the Regions of the Image that Contain Text (CRAFT)

CRAFT (Character-Region Awareness for Text detection) is an efficient and accurate text detection model that can detect text of different sizes, orientations, and fonts, it uses a CNN to produce two output maps: the Character region score map and the Affinity score map. Character region score map: is a binary map that indicates the likelihood of a pixel belonging to a character region, it is used to localize individual characters in the image. Affinity score map: is a binary map that indicates the likelihood of two adjacent characters belonging to the same text instance, it is used to group each character into a single instance(word). The generation and purpose of these maps are visually represented in Figure 3.2, which illustrates how CRAFT isolates characters and links them into meaningful text sequences.

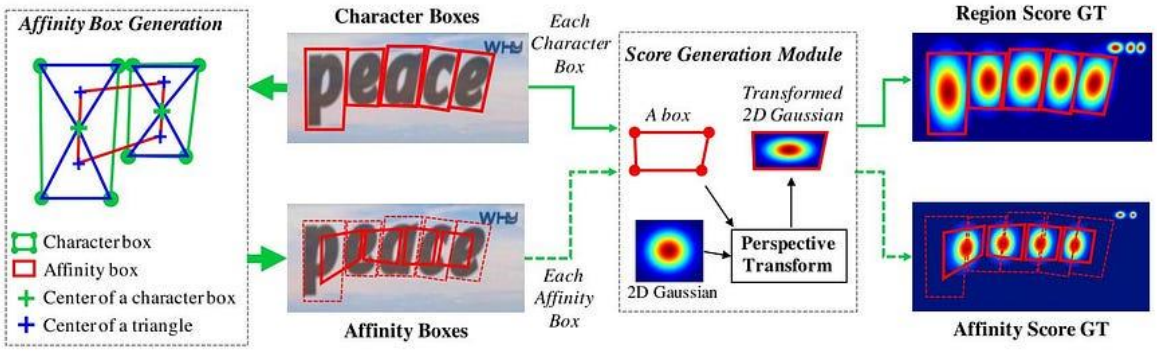


Figure 3.2 Character Region Score Map and the Affinity Score Map

3.4.2 The Convolutional Part of the Network (Feature Extraction)

In the CRNN model, the convolutional layers are constructed by adopting the convolutional and max-pooling layers from a standard CNN model, such as

ResNet© or VGG [4][17]. This convolutional component is used to ensure that a sequential feature representation is extracted from the input image. In other words, the image is processed in such a way that the spatial relationships between pixels are preserved while the dimensionality is reduced. These feature representations are then passed as input to the recurrent neural network (RNN) [4] component for further sequence modelling. The role of this convolutional block in CRNN are illustrated in Figure 3.3, highlighting how visual features are preserved and prepared for text recognition.

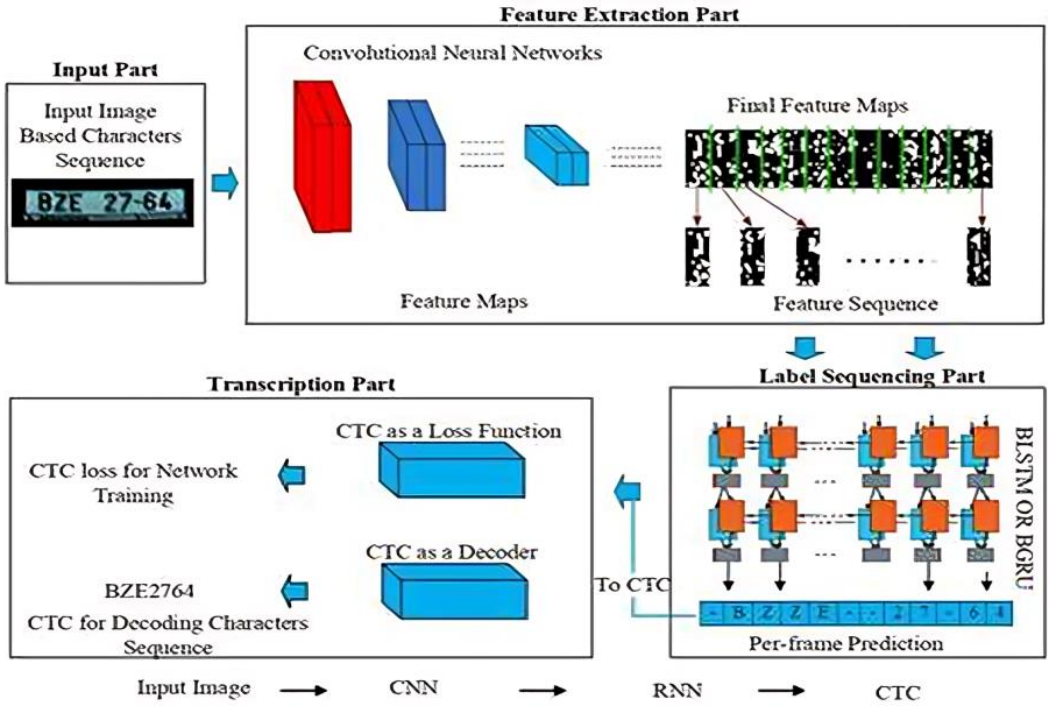


Figure 3.3 Convolutional Part of CRNN

3.4.3 The RNN Part of the Network

The RNN component of EasyOCR© is responsible for processing the sequential feature representation of the input image generated by the CNN component. As shown in Figure 3.4 it consists of multiple layers of Long Short-Term Memory (LSTM) cells that are designed to capture long-term dependencies in sequential data by selectively remembering and forgetting information. For example, consider a case where the CNN feature extraction layer produces an

encoding for “tha_k you” but the character at place “_” was not well recognized. Maybe there was a dirt spot at that place. But since the LSTM layers learned through data that there’s a vast amount of “thank you” inputs in the text, the OCR can predict the correct missing letter. The LSTM cells in the RNN component of EasyOCR are stacked together, with each layer feeding its output to the next layer as input. The final LSTM layer outputs a sequence of probabilities for each character in the input text. This output sequence is then passed through the (CTC) component to generate the final recognized text.

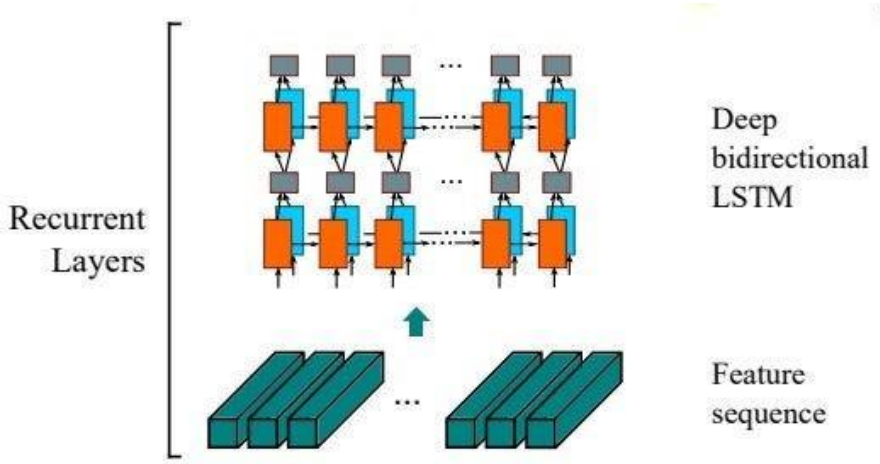


Figure 3.4 RNN a Part of CRNN

3.4.4 The CTC Component

CTC helps decode per-frame predictions made by RNN and transform it into a written text label sequence. The Sequence labelling problem consists of input sequences $X = [x1, x2, .., xT]$ and its corresponding output sequences $Y = [y1, y2, \dots, yU]$. In other words, this problem we’re dealing with involves accurately matching X to Y, but there are a couple of challenges to doing this. For one, both X and Y can be different lengths from each other. Additionally, we don’t have a clear and precise alignment of the elements in X and Y. The CTC algorithm overcomes these challenges. For a given X it gives us an output distribution over all possible Y’s. We can use this distribution either to infer a likely output or to assess the probability of a given output. For instance, in Figure 3.5, the sequence

"TANN" is predicted, but CTC allows merging of duplicate letters while removing blank tokens ("-"), resulting in the final output: "TAN"

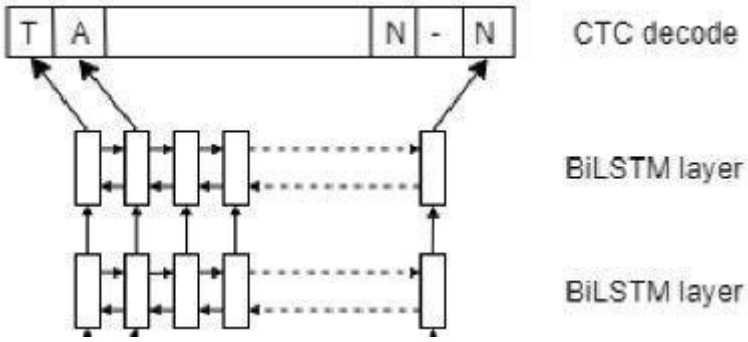


Figure 3.5 CTC Component

In this stage, the system determines whether any meaningful text has been successfully extracted from the image. If no text is found, the system returns to the "Capture Frame from Webcam" block and retries with a new frame. This ensures that the system does not proceed with an empty or incorrectly processed text input. If text is detected, it moves forward to analyse the language. As shown in the Table 3.3, no resizing occurs at any stage, so the image remains 640x480 pixels throughout the pipeline.

S.NO	Stage	Image Size
1	Webcam Capture	640x480 (RGB)
2	Grayscale Conversion	640x480 (Gray)
3	CLAHE Processing	640x480 (Gray)
4	Gaussian Blur	640x480 (Gray)
5	OCR Processing	640x480 (Gray)

Table 3.3 Summary of Image Sizes at Different Stages of Processing

3.5 LANGUAGE DETECTION (ENGLISH, TELUGU & KANNADA)

After extracting text, the system identifies the language. Since it supports multiple languages (English, Telugu, and Kannada), the OCR engine classifies

the extracted text based on linguistic patterns, character structures, and word formations. Detecting the correct language is crucial for further steps, especially if the text needs to be translated before conversion to speech.

A key aspect of language detection in our project is the use of Unicode character ranges, which help in differentiating text from various languages based on their unique script encoding. Unicode is a standardized encoding system that assigns a unique code to every character in all writing systems, enabling seamless text processing across different languages. In our implementation, we use Unicode character ranges to classify the detected text by checking the character codes within predefined script ranges. For example, in our project, we define language detection based on the following Unicode ranges

```
self.script_ranges = {
    'te': (0x0C00, 0x0C7F), # Telugu characters
    'kn': (0x0C80, 0x0CFF), # Kannada characters
    'en': (0x0000, 0x007F) # English characters (ASCII range) }
```

Each character in the extracted text is checked against these ranges. If the character falls within the 0x0C00 to 0x0C7F range, it is identified as Telugu if it falls within 0x0C80 to 0x0CFF, it is recognized as Kannada and if it belongs to the 0x0000 to 0x007F range, it is classified as English. By analysing the frequency of characters in each range, the system determines the dominant language of the text.

Examples of Unicode Characters in Telugu and Kannada

To better understand how Unicode helps in detecting languages, here are some examples

- Telugu Character Example
    - Character: అ (First letter of Telugu alphabet)
    - Unicode: U+0C05 (Falls within the Telugu script range: 0x0C00 - 0x0C7F)
- Kannada Character Example
    - Character: ಅ (First letter of Kannada alphabet)
    - Unicode: U+0C85 (Falls within the Kannada script range: 0x0C80 - 0x0CFF)
- English Character Example
    - Character: A
    - Unicode: U+0041 (Falls within the ASCII range: 0x0000 - 0x007F)

This approach is highly effective because it enables the system to detect multilingual content dynamically, even when text from different languages appears in the same image. It ensures that the correct language is selected before further processing, allowing accurate translation and speech synthesis. By integrating Unicode-based script detection, our system enhances its ability to support multiple languages efficiently, providing a seamless and accessible experience for visually impaired users.

Next stage is to look for Language conversion, is the process of translating text from one language to another while preserving its meaning. Unlike transliteration, which retains pronunciation, language conversion focuses on accurately conveying the message in the target language. This step is essential when the extracted text is not in English and needs translation before speech synthesis. For example

- If the detected text is in English, the system bypasses translation and directly converts it to speech.

- If the detected text is in Kannada, Telugu, the system translates it into phonetic language before proceeding to speech synthesis.

This decision-making step ensures efficient processing by avoiding unnecessary conversions. If translation is required, the system sends the extracted text through a language conversion module, which translates it into English before passing it to the Text-to-Speech (TTS) engine. If the text is already in English, the system skips translation and proceeds directly to TTS processing, ensuring a faster and seamless user experience.

3.6 TRANSLITERATION (RULE - BASED WORKFLOW)

Transliteration is the process of converting text from one script to another while maintaining the pronunciation as closely as possible. Unlike translation, which conveys meaning, transliteration focuses on phonetic representation. For example, the Hindi word "नमस्ते" is transliterated into English as "Namaste" but not translated.

Some more examples,

1. Hello → Telugu: హలో (Halo), Kannada: ಹಲೋ (Halo)
2. Welcome → Telugu: వెల్‌కమ్ (Velkam), Kannada: ವೆಲ್ಕಮ್ (Velkam)
3. Thank You → Telugu: థ్యాంక్ యూ (Thyāṅk Yū), Kannada: ಥ್ಯಾಂಕ್ ಯು (Thyāṅk Yū)
4. Good morning → Telugu: గుడ్ మార్నింగ్ (Guḍ Mār̲niṅg), Kannada: ಗುಡ್ ಮೋರ್ನಿಂಗ್ (Guḍ Mōrniṅg)
5. India → Telugu: ఇండియా (Iṇḍiyā), Kannada: ಇಂಡಿಯಾ (Iṇḍiyā)
6. Water → Telugu: వాటర్ (Wāṭar), Kannada: ವಾಟರ್ (Wāṭar)

7. School → Telugu: స్కూల్ (Skūl), Kannada: ಸ್ಕೂಲ್ (Skūl)
8. Computer → Telugu: కంప్యూటర్ (Kaṃpyūṭar), Kannada: ಕಂಪ್ಯೂಟರ್ (Kaṃpyūṭar)
9. Love → Telugu: లవ్ (Lav), Kannada: ಲವ್ (Lav)
10. Happy → Telugu: హ్యాపీ (Hyāpī), Kannada: ಹ್ಯಾಪಿ (Hyāpi)

3.6.1 Rule-Based Workflow for Transliteration

Transliteration is often done using predefined rules or machine learning models that consider linguistic patterns. Some common approaches include

1. Rule-Based Systems – Using manually crafted mappings between scripts (e.g., ISO 15919 for Indic scripts).
2. Statistical Models – Leveraging probability-based approaches to determine the best character mappings.
3. Neural Network Models – Using deep learning techniques, such as transformer models, for context-aware transliteration.

In our project we use the rule-based workflow for transliteration referred from [6] adheres to pre-defined linguistic rules and mapping tables to transform text from one script to another. Rule-based systems have a number of advantages over statistical or AI-based systems.

A rule-based workflow for transliteration uses a pre-defined set of rules and mapping tables to transliterate text from one script to another with phonetic accuracy. This is a deterministic process, where the same input will always generate the same output. A script detection module identifies the dominant language of the recognized text based on Unicode character ranges, as shown in Figure 3.6. The Indic transliteration [6] library uses deterministic rule-based

mappings to convert Indic scripts (e.g., Telugu) to ITRANS, a Romanized phonetic scheme. The process involves script segmentation, grapheme-to-phoneme mapping [4], and contextual postprocessing to ensure proper pronunciation.

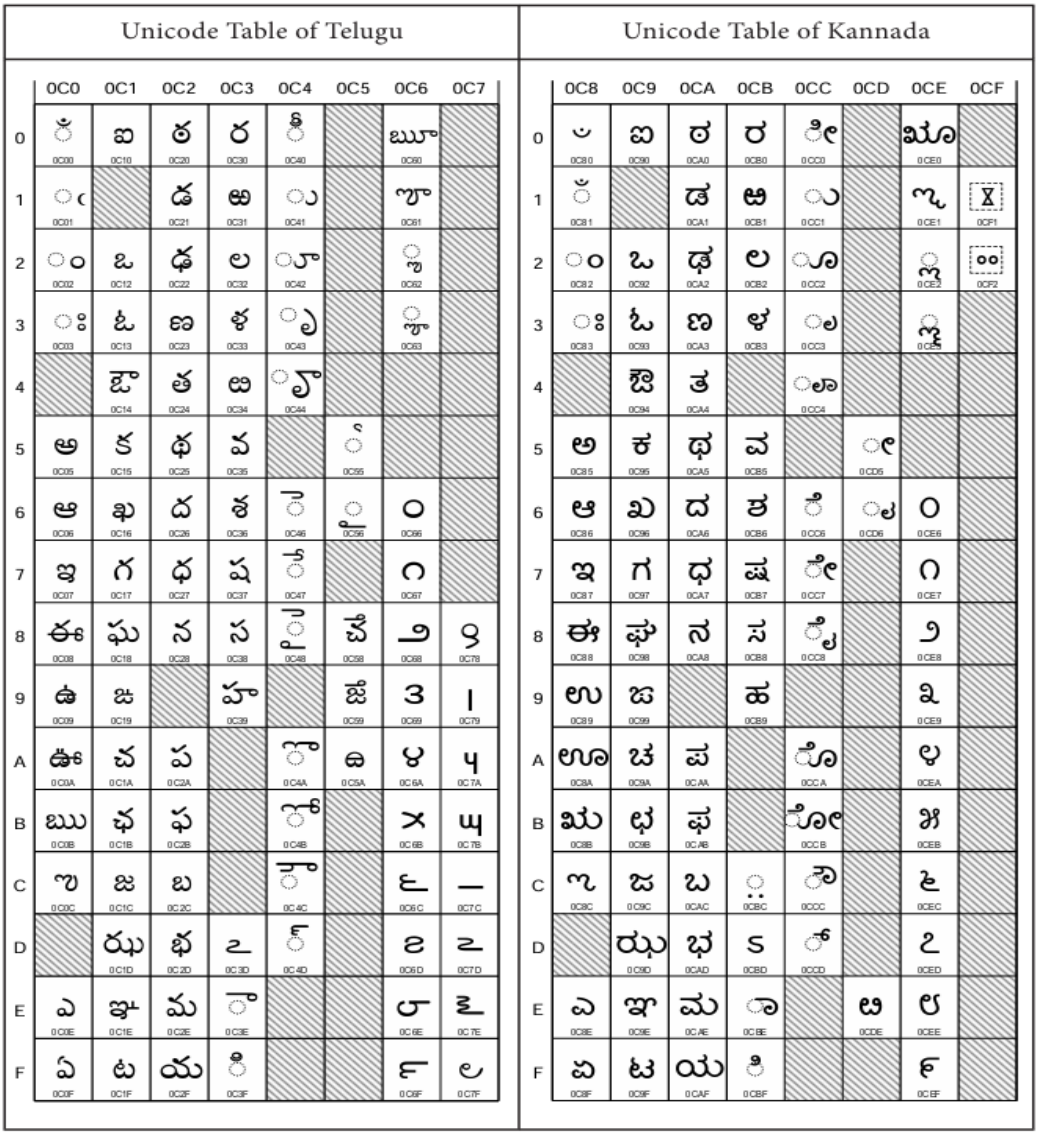


Figure 3.6 Unicode Character for Telugu and Kannada.

Figure 3.6 illustrates the Unicode Character Table for Telugu and Kannada, which is used for script detection and transliteration. The figure provides the Unicode ranges for each character, enabling the system to accurately identify and process Indic scripts. A rule-based transliteration system generally operates in these five main steps

Step 1: Input Processing

The input message is taken in one script (e.g., English, Latin script) and must be transformed into another script (e.g., Telugu, Kannada). The message is tokenized (broken up into characters, words, or syllables) for simpler handling.

Step 2: Character Mapping & Substitution Rules

Every character or syllable of the input script is associated with its relative character/sound in the destination script through an existing character mapping table. Telugu Transliteration example mapping

ka → క, kha → ఖ, ga → గ, tha → థ, ma → మ, ra → ర

sh → శ, ch → చ, ee → ఈ, oo → ఊ

For Kannada

ka → ಕ, kha → ಖ, ga → ಗ, tha → ಥ, ma → ಮ, ra → ರ

sh → ಶ, ch → ಚ, ee → ಈ, oo → ಊ

Step 3: Phonetic Adjustments

There are several different ways of pronouncing a sound in some languages. The system uses phonetic rules to provide proper pronunciation. Example "ai" in English is generally pronounced as "ஐ" in Tamil or "ಐ" in Kannada.

Step 4: Context-Based Modifications

Some letters vary depending on their position within a word. Example (Telugu) ka (క) + sha (ష) should be merged into ksha (క్ష) rather than కష. ra

(ర) in the middle of a word might sometimes have to be altered depending on context.

Step 5: Output Generation

The resulting transliterated text is constructed using the mapped characters and phonetic rules. The text is then normalized so that special symbols and diacritics are properly applied

Telugu Input	Expected Phonetic Output (English)	Indic Transliteration Output
నమస్తే	namaste	namaste
గురు	guru	guruu
రామ	rama	rama
శివ	shiva	siva
కృష్ణ	krishna	krsna

Table 3.4 Telugu to Phonetic English Transliteration

Kannada Input	Expected Phonetic Output (English)	Indic Transliteration Output
ನಮಸ್ತೆ	namaste	namaste
ಗುರು	guru	guruu
ರಾಮ	rama	rama
ಶಿವ	shiva	siva
ಕೃಷ್ಣ	krishna	krsna

Table 3.5 Kannada to Phonetic English Transliteration

Tables 3.4 and 3.5 shows the precision of transliteration for Telugu and Kannada text into English phonology. The data indicate that while most words are transliterated correctly, some phonetic representations deviate from conventional spellings. For example, నమస్తే (Namaste) and రాము (Rama) are accurately transliterated, whereas శివ (Shiva) and ಶಿವ (Shiva) appear as *siva* due to the scheme's inconsistency in distinguishing *sh* and *s*. Similarly, కృష్ణ (Krishna) and ಕೃಷ್ಣ (Krishna) are transliterated as *krsna*, following a structured phonetic coding but deviating from common English usage. These variations suggest that while the transliteration scheme is systematic and rule-based, certain refinements may improve its alignment with intuitive phonetic spellings.

This transliteration model has a number of significant benefits like fast and efficient, and completely offline, which is useful in settings with limited internet access. The output is consistent, which makes it easy to debug and maintain, and increases the model's reliability. It does not rely on large datasets in low-resource languages converting to Roman Scripts, which is useful in low-resource languages. Finally, its lightweight design allows for a quicker deployment especially in specialized or domain-related use cases.

CHAPTER 4

PROPOSED METHOD

4.1 SCOPE OF WORK

The solution is to create a low-cost, offline, and highly flexible voice-enabling text reader that facilitates visually impaired people to read and understand printed or handwritten text independently. With the use of Optical Character Recognition (OCR) and Text-to-Speech (TTS) technology, the system allows real-time text identification and voice output without internet access. The solution solves the issue of the expensiveness and restriction of available services and provides a trustful and convenient means for reading text-based information in many languages.

4.2 PROJECT DESCRIPTION

The goal of the proposed solution is to ensure assistance is provided to individuals with vision challenges by enabling the conversion of handwritten or printed text into synthesized speech that can be heard by the user. In contemporary society, an abundance of written text and documents is conveyed through various materials and forms such as books, printed content, billboards, product labels, and official documentation. Unfortunately, independent access to this information is often hindered for individuals with impaired sight. A method has been developed using a webcam through which the text material can be captured from its source, allowing the scanning of text from books, documents, or other surfaces containing printed content. Independence in accessing physical textual content is promoted by this real-time capture strategy, with improvements made to user experience through accessibility enhancements and publicly available features.

Once the text has been captured, it is extracted from the image through Optical Character Recognition (OCR) technology. This stage is considered critical as the visual appearance of the text is transformed into electronic,

machine-readable characters that can be further processed. Language detection is then applied to the extracted text to identify whether it is in Kannada, Telugu, or English. An automated language conversion feature is offered to enable translation of the text into other languages, ensuring comprehension regardless of the original language. The usability of the system is increased, and greater access is facilitated through its multi-language capabilities.

In the final stage, the text is converted into speech using a Text-to-Speech (TTS) engine. This solution is found to be beneficial for individuals with vision impairments who prefer to listen to text in a voice that is both intelligible and natural-sounding. Comprehension is improved through this method, and fluid speech is provided to simplify following the content. After the speech is generated, OCR is again performed to capture and process new text for further interaction. This end-to-end solution comprising OCR, language detection, translation, and TT is implemented as a fully assisted system. Not only is empowerment provided to individuals with visual impairments to read surrounding text, but enhanced engagement is also enabled with educational content, official documents, and everyday writings without constant dependence on others.

4.3 TEXT RECOGNITION AND SPEECH CONVERSION PROCESS

The Text Recognition and Speech Conversion Process starts with a webcam or camera module capturing an image containing handwritten or printed text. The rendered image is the first step for the entire system. After the image has been captured, the image is pre-processed to improve the clarity and contrast of the text for better recognition. The pre-processed image is sent through an Optical Character Recognition (OCR) engine, such as EasyOCR©, which identifies the text contained in the image, captures and extracts it from the image,

and converts it to machine-readable format. After the OCR stage, language detection and language correction (if desired) may be applied to improve accuracy and readability, especially in multi-language text, e.g., Kannada to English or Telugu to English. The final stage of the process involves sending the cleaned-up detected text to a Text-to-Speech (TTS) Engine. The TTS generates an audible voice output of clean, natural-sounding speech. The purpose of the process is to allow users to hear cleaned recognizable text and perfectly understand that text through audible speech delivery. This reading process is designed to operate in real-time and is user-friendly for user experience.

4.3.1 System Initialization Phase

When powered on, the system is initialized by loading necessary software libraries, verifying hardware components like the webcam and speakers, and preparing for user interaction.

4.3.2 Webcam and OCR Engine Initialization

This first step guarantees that all components are ready to run in a continuous loop. Once the initial setup is complete, the camera will capture real-time images of the surrounding text and EasyOCR© will begin to load into memory to read the text. Properly synchronizing the camera and the OCR engine avoids issues or delays later on.

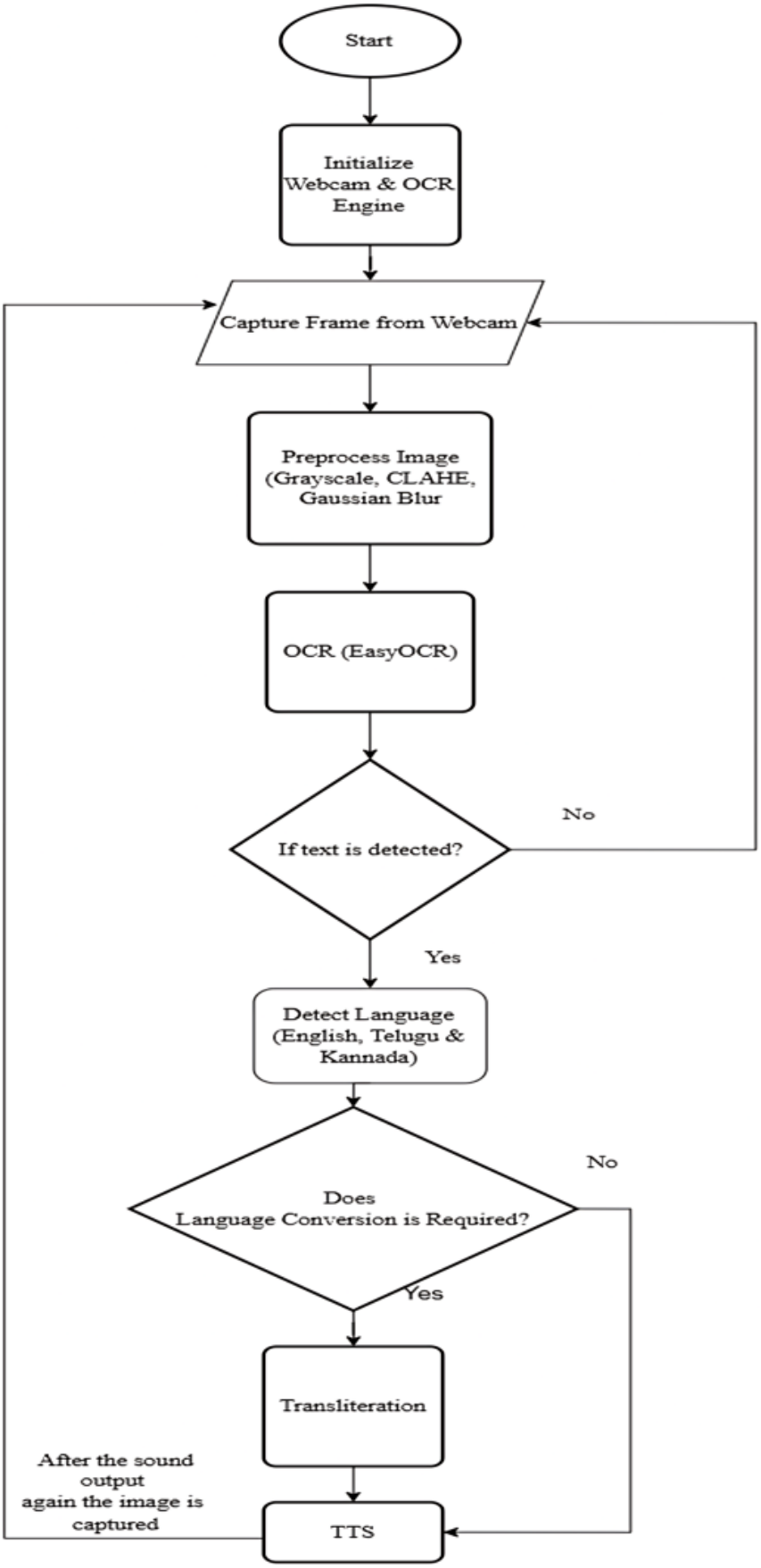


Figure 4.1 Illustration of the Proposed Method

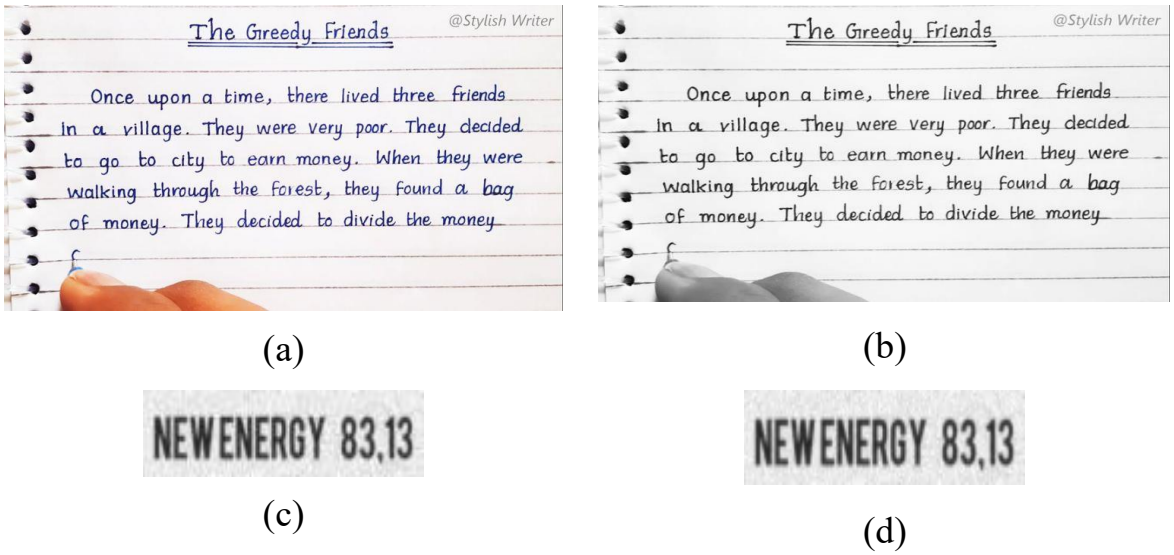
4.3.3 Frame Capture Process

At this stage, the system captures an image frame of resolution 640x480 pixels using the webcam. This frame acts as an input containing the printed or handwritten text that needs to be processed. The webcam continuously takes new frames in real time, ensuring that users can point the camera at different documents, signboards, or objects containing text without manually restarting the system. If the frame captured is of poor quality or blurry, the system ensures multiple attempts to improve recognition accuracy.

4.3.4 Preprocessing Input Image

4.3.4.1 Grayscale Conversion

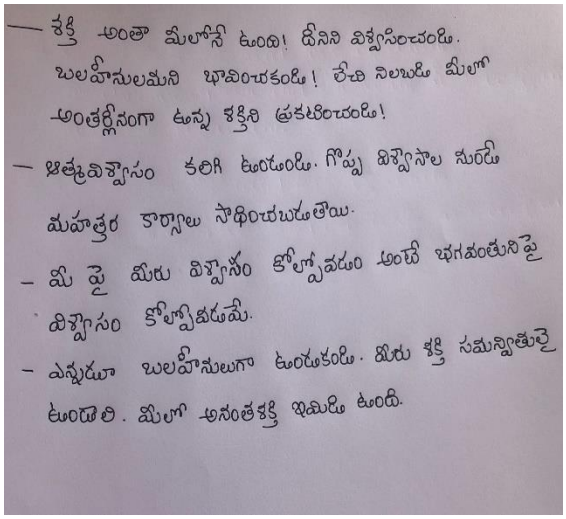
The image preprocessing and enhancement function plays a critical role in preparing images for accurate text recognition. Initially, the image is converted from its original colour format to grayscale as given in Figure 4.2, simplifying the data by reducing it to intensity values. This helps in focusing the analysis purely on the shape and structure of the text rather than the colour, which is generally not useful in OCR tasks.



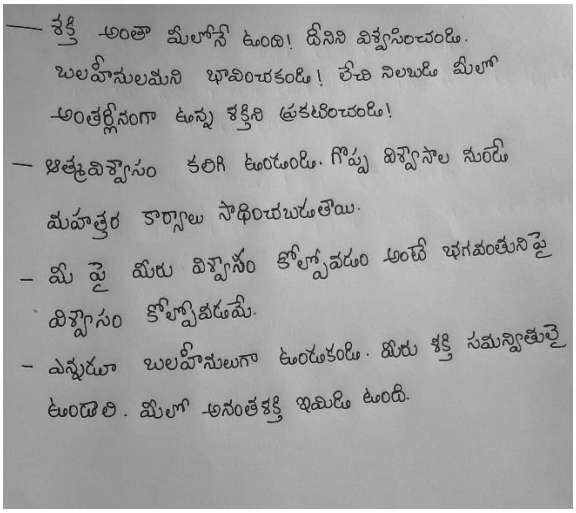
(a) (b)

(c) (d)

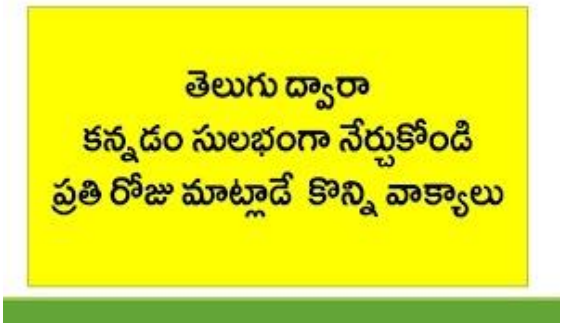
Figure 4.2 RGB to Grayscale Conversion of Sample Images with English Text



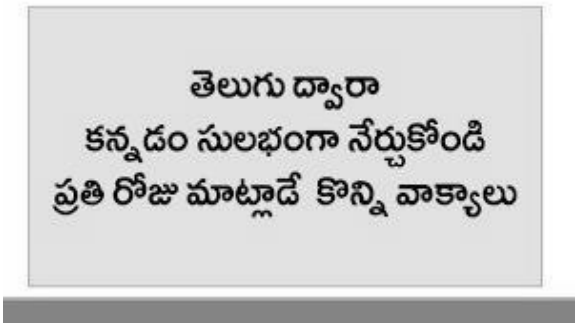
(a)



(b)

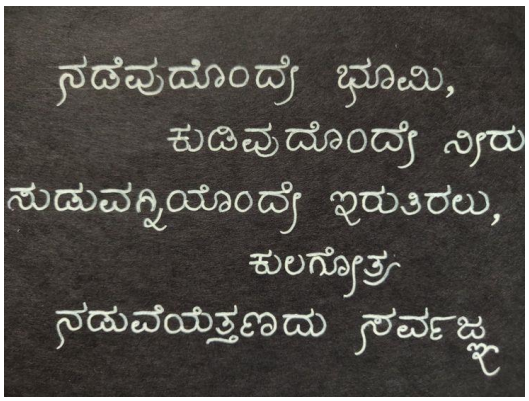


(c)

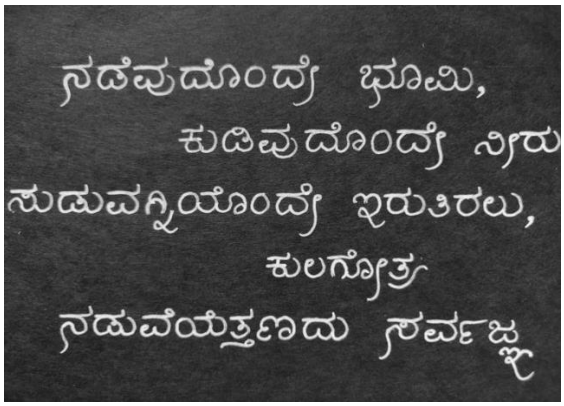


(d)

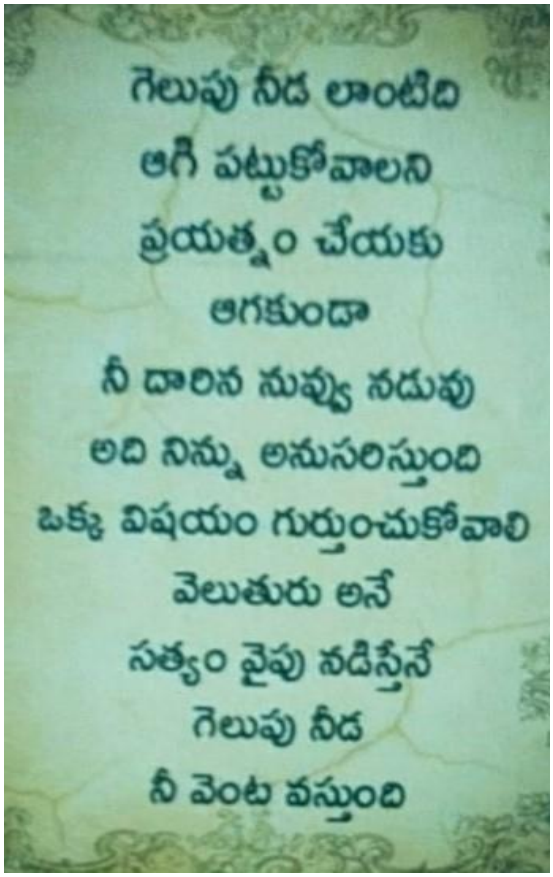
Figure 4.3 RGB to Grayscale Conversion of Sample Images with Telugu Text



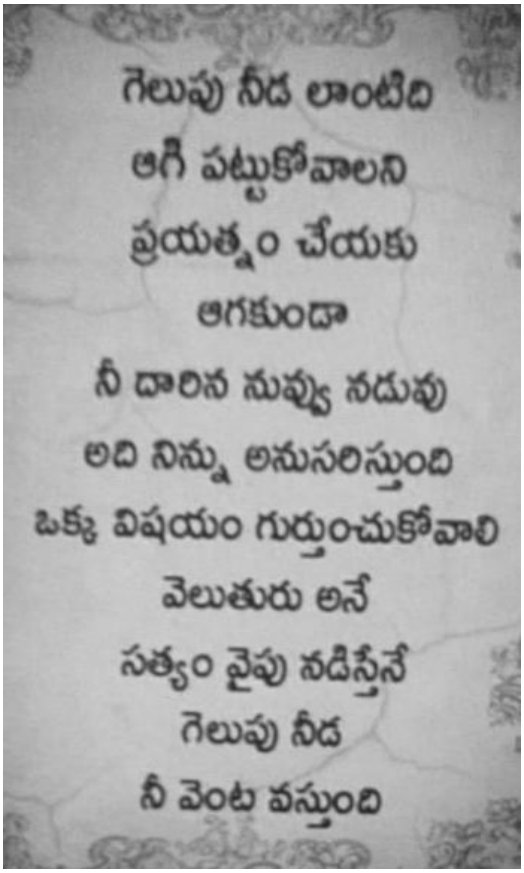
(a)



(b)



(c)



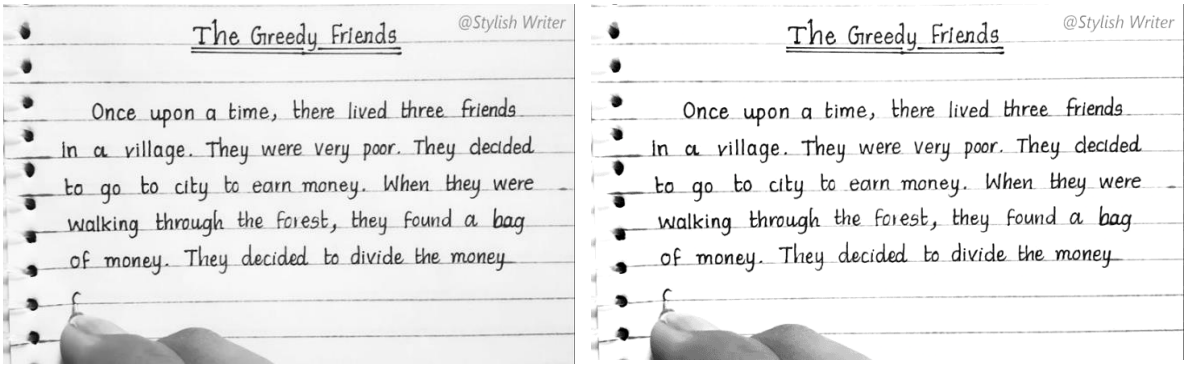
(d)

Figure 4.4 RGB to Grayscale Conversion of Sample Images with Kannada Text

4.3.4.2 CLAHE (Contrast Limited Adaptive Histogram Equalization)

Following this, Contrast Limited Adaptive Histogram Equalization (CLAHE) is applied. CLAHE© enhances the contrast of the image locally, dividing it into small tiles and improving the visibility of text within each region. This technique is particularly effective for images with uneven lighting or shadows, ensuring that text in darker or brighter regions becomes more distinguishable. To further refine the image, Gaussian Blur is introduced. This technique smoothens the image by averaging pixel values with their neighbours, reducing small noise and irrelevant details while preserving the edges of text

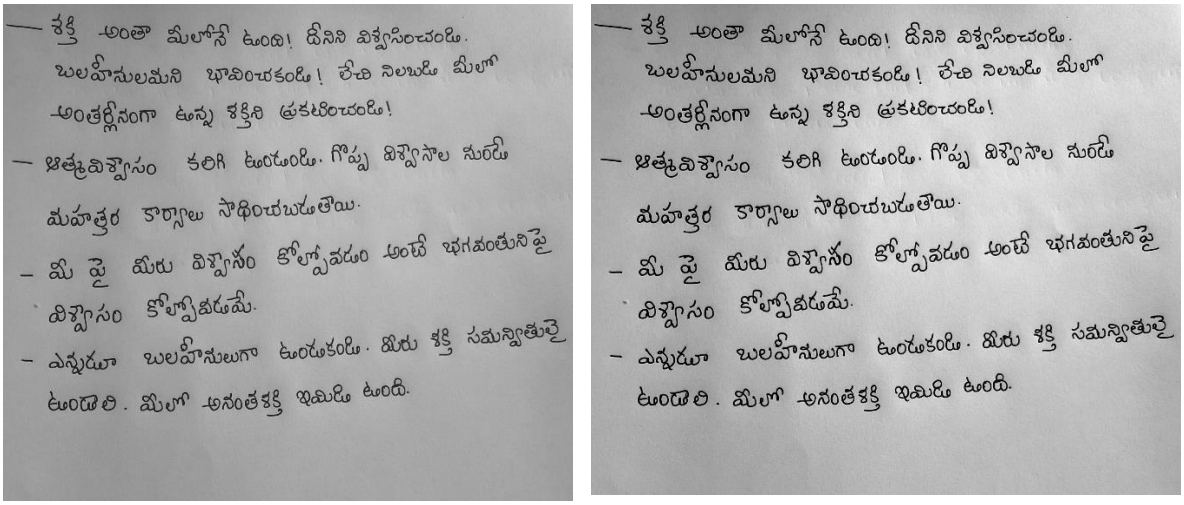
characters. As a result, the final processed image becomes cleaner and more uniform, enabling OCR engines to detect and recognize text more accurately.



(a) (b)

(c) (d)

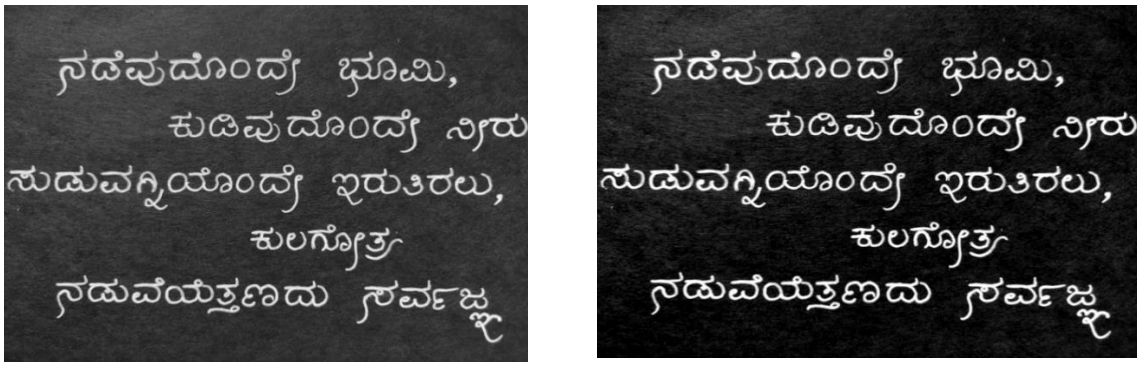
Figure 4.5 Grayscale Image to Contrast Enhanced and Noise Reduced Image of Sample Images with English Text



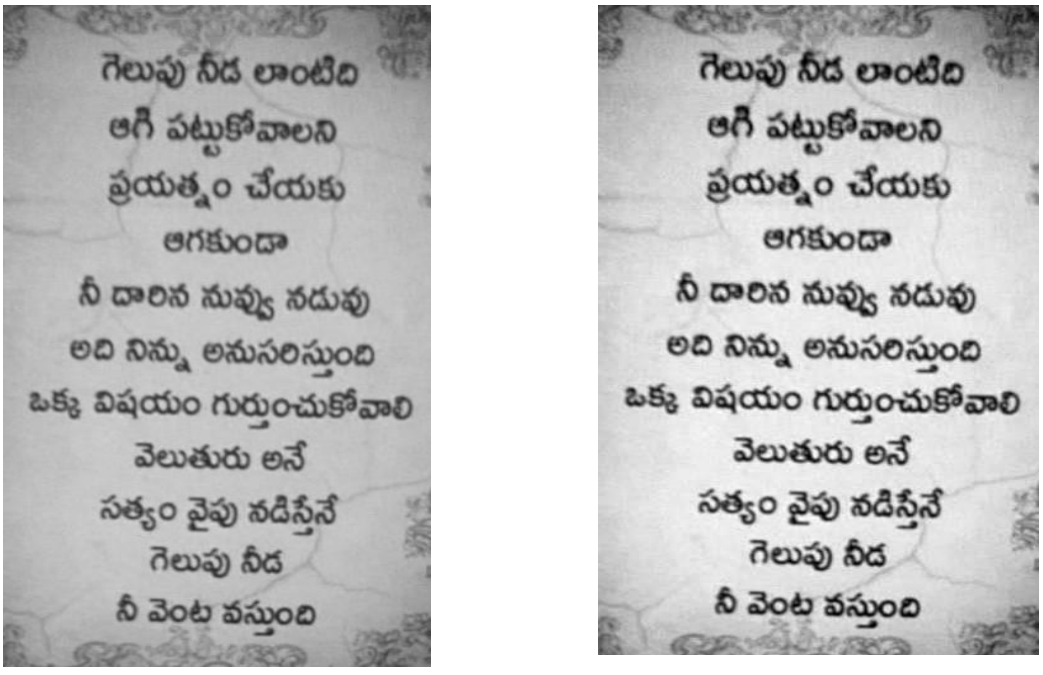
(a) (b)

(c) (d)

Figure 4.6 Grayscale Image to Contrast Enhanced and Noise Reduced Image of Sample Images with Telugu Text



(a) (b)

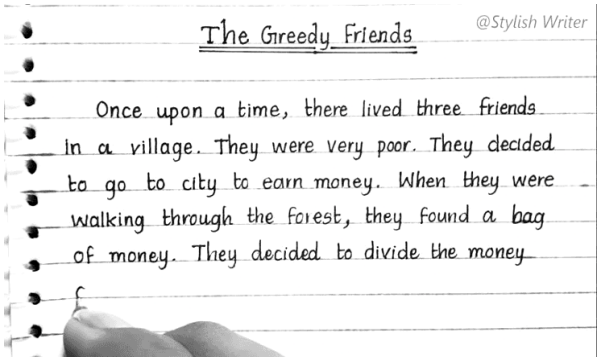


(c) (d)

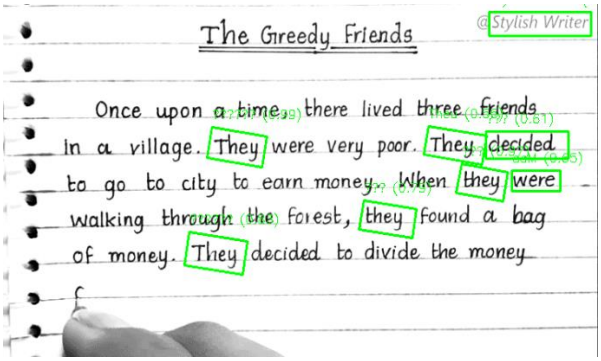
Figure 4.7 Grayscale Image to Contrast Enhanced and Noise Reduced Images of Sample Image with Kannada Text

4.3.5 Text Extraction

Text detection in an image involves identifying regions that likely contain textual content. This is typically done using deep learning models like CRAFT (Character Region Awareness for Text detection), which analyse the image to generate two key outputs: character region scores and affinity scores. The character region scores map highlights areas that resemble individual characters, while the affinity score map helps link nearby characters that belong to the same word. These maps are combined to form bounding boxes around text areas, accurately detecting text regardless of size, font, or orientation in the image.



(a)

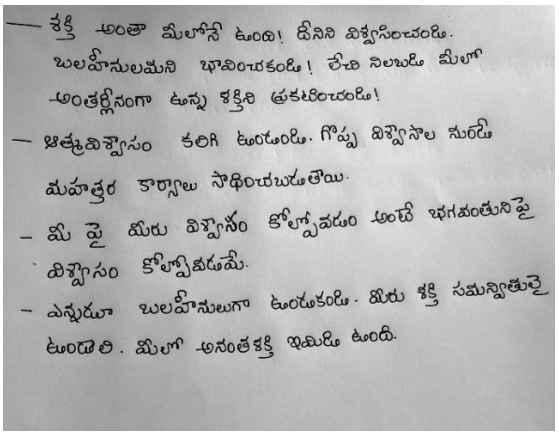


(b)

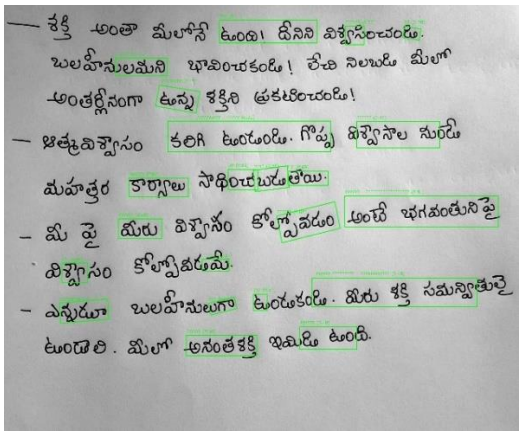
(c)

(d)

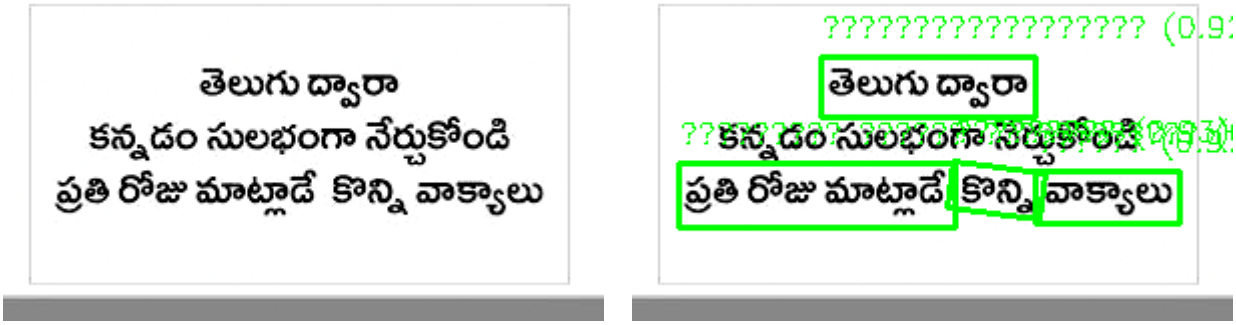
Figure 4.8 Extracted Text from Sample Images with English Text



(a)

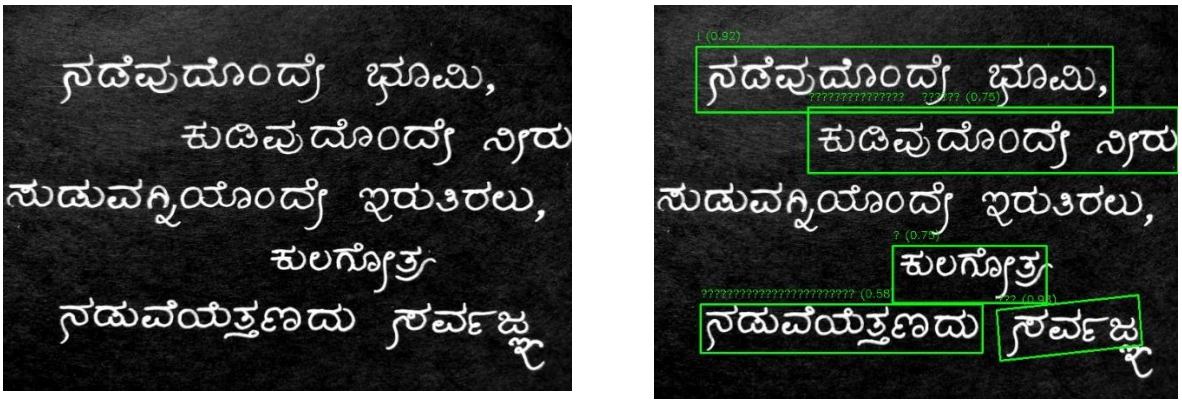


(b)

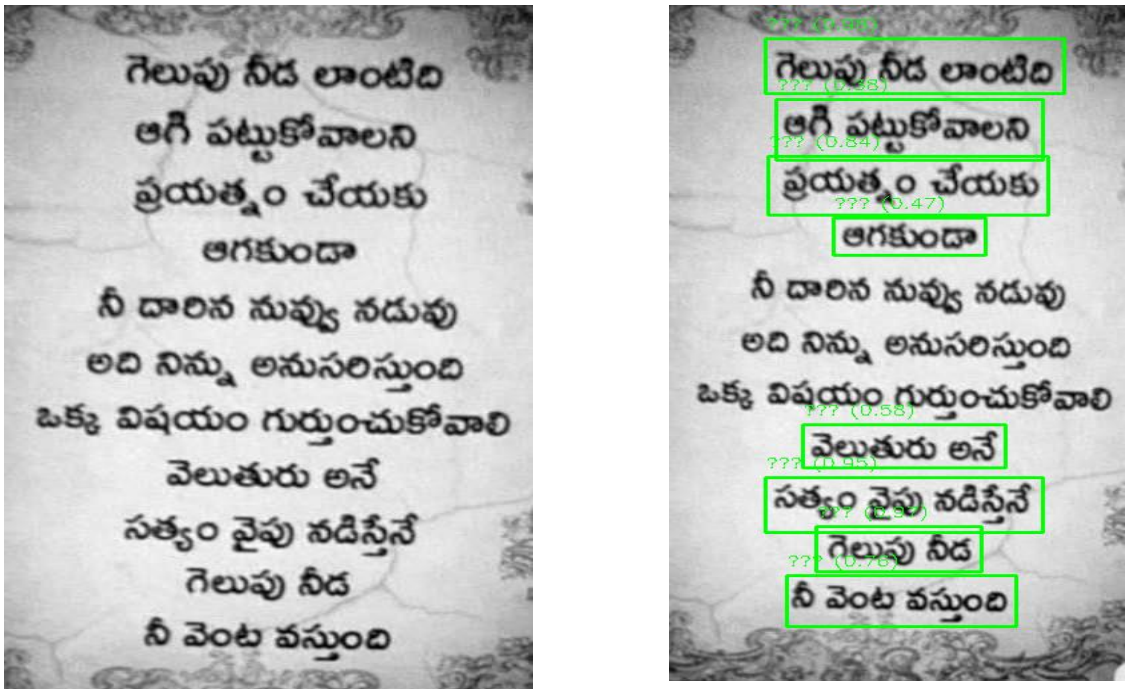


(c) (d)

Figure 4.9 Extracted Text from Sample Images with Telugu Text



(a) (b)



(c) (d)

Figure 4.10 Extracted Text from Sample Images with Kannada Text

Language	Image	Accuracy Range (0–1)	CPU Processing Time (seconds)
English	Handwritten Text	0.58 – 0.78	1.8 – 2.5
	Printed Text	0.90 – 0.98	0.6 – 1.2
Telugu	Handwritten text	0.55 – 0.76	2 – 2.7
	Printed text	0.85 – 0.93	2.1 – 2.8
Kannada	Printed text	0.76 – 0.94	1.1 – 1.7

Table 4.1 Accuracy and CPU Processing Time

Table 4.1 illustrates the OCR accuracy and CPU processing time of EasyOCR© for English, Telugu, and Kannada languages on Raspberry Pi 5© (8GB RAM). The accuracy is represented in a normalized range between 0 and 1, indicating the confidence level of text predictions. English printed text generally achieves higher accuracy due to the model's extensive training on Latin script datasets, whereas Telugu and Kannada, being complex scripts, may yield slightly lower scores depending on font style and clarity. The CPU processing time reflects the duration required to perform inference per image, which varies based on language complexity, image resolution, and script density. Despite limited computational resources, EasyOCR demonstrates reliable performance across all three languages, making it suitable for lightweight OCR tasks in multilingual applications.

4.3.6 Fundamentals of TTS (Text-to-Speech)

Once the text is fully processed, it is passed to the Text-to-Speech (TTS) engine, which plays a crucial role in converting the extracted text into spoken words. The TTS engine is designed to ensure that the speech output is natural and clear, incorporating human-like intonation and rhythm for better comprehension. Unlike traditional robotic-sounding speech synthesis, this system is optimized to produce continuous and smooth narration, avoiding abrupt pauses or unnatural

breaks that might disrupt the listening experience. Another critical aspect of the TTS engine is its adaptability to different languages, ensuring that words are pronounced correctly based on their linguistic origin. If the text was originally in English, it is directly converted to speech, whereas Telugu and Kannada text undergo transliteration before being synthesized, allowing the system to maintain proper pronunciation and fluency in the generated speech. Once the audio is produced, it is played through the speakers, enabling visually impaired users to effortlessly listen to the extracted text without any additional effort.

4.3.7 Looping mechanism

This system incorporates a loop that reads text continuously, and, after it generates its audio output, it will return to the "Capture Frame from Webcam" step. This means that new text can also be scanned seamlessly without further user intervention. This is a great feature for reading books, signboards, documents, etc.

CHAPTER 5

SYSTEM SPECIFICATION

Our proposed system is built on a Raspberry Pi© platform, which serves as the core processing unit. The setup integrates hardware components, software packages, and a structured workflow to achieve real-time text-to-speech conversion. Below is a detailed description of the system specification:

5.1 HARDWARE DESCRIPTION

5.1.1 Raspberry Pi Architecture - An Overview

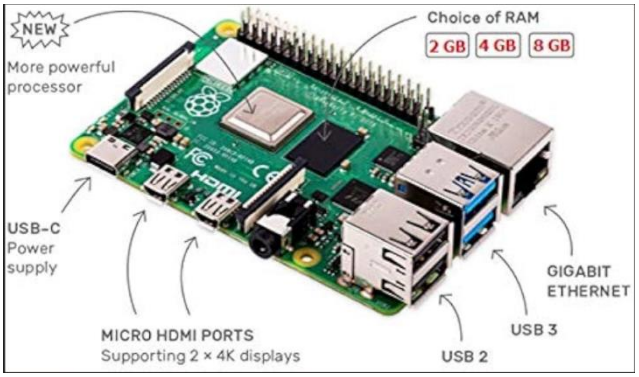


Figure 5.1 Raspberry Pi 5

The Raspberry Pi© serves as the Central Processing Unit (CPU) for the system. It is a compact, single-board computer that is widely used in embedded systems, IoT projects, and prototyping due to its versatility, affordability, and low power consumption. In this project, the Raspberry Pi© is responsible for running the OCR (Optical Character Recognition), text processing, and TTS (Text-to-Speech) modules, making it the core component of the system. The specific model used in this project is the Raspberry Pi 5 Model B with 8GB RAM©. This model is chosen for its enhanced computational power, which is sufficient for real-time processing tasks.

Key features of the Raspberry Pi 5 Model B include

- A 2.0 GHz quad-core ARM Cortex-A76 CPU for high-performance processing, providing better performance compared to the previous Raspberry Pi 4.
- 8GB LPDDR4X RAM for efficient multitasking and handling memory-intensive applications (LPDDR4X RAM is a slightly upgraded version from the previous LPDDR4 in the Pi 4).
- Dual-band Wi-Fi (2.4 GHz and 5 GHz) and Bluetooth 5.0 for wireless connectivity, enabling fast data transfer and communication with peripherals.
- USB 3.0 ports and an additional USB 2.0 port, supporting high-speed data transfer.
- Gigabit Ethernet for wired network connectivity.
- Two Micro-HDMI ports supporting dual 4K display output (4K @ 60Hz per port).
- 40-pin GPIO header for interfacing with external hardware components.
- Improved cooling capabilities, with support for higher temperatures and better power efficiency, making it more suitable for industrial applications.

GPIO Pins and Functions – Raspberry Pi 5

The Raspberry Pi 5© provides a 40-pin GPIO header for interfacing with external hardware like sensors and actuators.

- Power Pins
    - 3.3V & 5V – Provide power to peripherals.
    - GND (Ground) – Reference ground for circuits.

- GPIO Pins
    - Configurable as input/output for digital communication.
    - Support I2C, SPI, and UART protocols.
- Special-Purpose Pins
    - I2C (SDA, SCL) – For I2C communication.
    - SPI (MOSI, MISO, SCLK, CE) – For SPI communication.
    - UART (TX, RX) – For serial communication.
- PWM Pins
    - Control motors, LEDs, and other PWM-based devices.

5.1.2 Setup of Raspberry Pi 5

Here's a step-by-step guide to get Raspberry Pi up and running

The Raspberry Pi 5 Model B© includes updated drivers and supports faster communication with advanced peripherals, enhancing its versatility for prototyping and development. To set up the Raspberry Pi©, first, insert a microSD card with the operating system into the SD card slot (installation of OS is explained in 4.1.2). Next, connect a monitor using an HDMI cable and plug in a USB keyboard and mouse into the available USB ports. Additionally, connect a USB webcam for video input as per the Figure 4.2. Once all peripherals are connected, power on the Raspberry Pi© by plugging in the power adapter. The system will boot up, displaying the interface on the monitor, allowing you to interact with the device. After setting up the OS and connection of the components, follow these steps to ensure all peripherals are properly connected and recognized by the Raspberry Pi©.

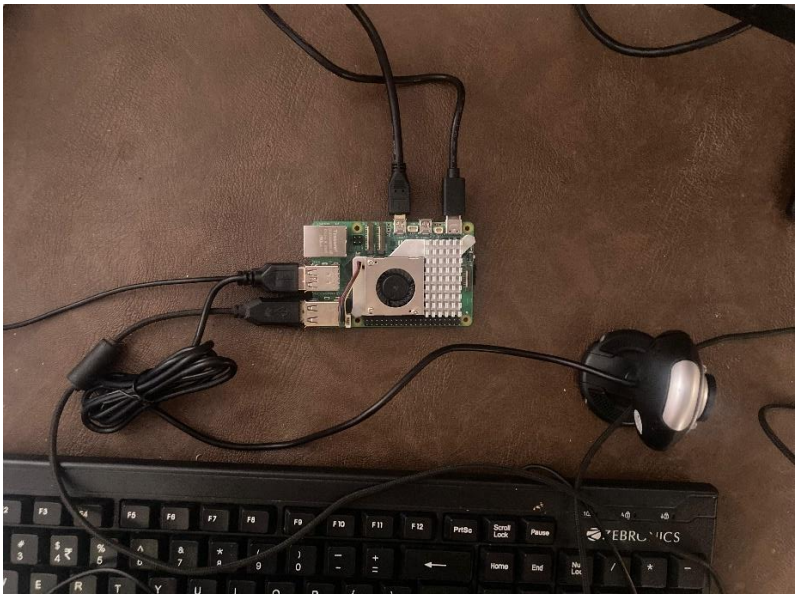


Figure 5.2 Hardware Setup

5.2 SOFTWARE DESCRIPTION

The proposed system requires a set of essential software components to ensure smooth functioning across image processing, OCR, language detection, and Text-to-Speech synthesis. Firstly, Python serves as the core programming language due to its rich ecosystem of libraries and compatibility with Raspberry Pi. OpenCV is used for image preprocessing tasks such as grayscale conversion, CLAHE©, and blurring. EasyOCR©, a deep learning-based optical character recognition library, is employed for extracting text from images in multiple languages including English, Telugu, and Kannada. For language detection and transliteration, libraries like langdetect and indic-transliteration are used. Pyttsx3 is integrated as an offline Text-to-Speech (TTS) engine to convert the extracted text into speech output. Additionally, essential dependencies such as NumPy and torch (for EasyOCR) must be installed. These software tools collectively ensure the end-to-end functionality of the system from image capture to audio output.

CHAPTER 6

RESULTS AND DISCUSSION

This section analyses the results of the proposed method, focusing on accuracy, efficiency, and performance. A comparison with existing techniques highlights improvements and challenges, while factors affecting accuracy, processing speed, and transliteration quality are discussed. The system was tested with various images and the accuracy were evaluated, with EasyOCR© demonstrating superior performance in multilingual text recognition.

The captured text undergoes preprocessing and recognition, with sample input images and processed outputs showcasing the system's effectiveness. The results confirm that the proposed method achieves high accuracy and real-time processing capabilities, making it a viable solution for multilingual text recognition and speech synthesis.

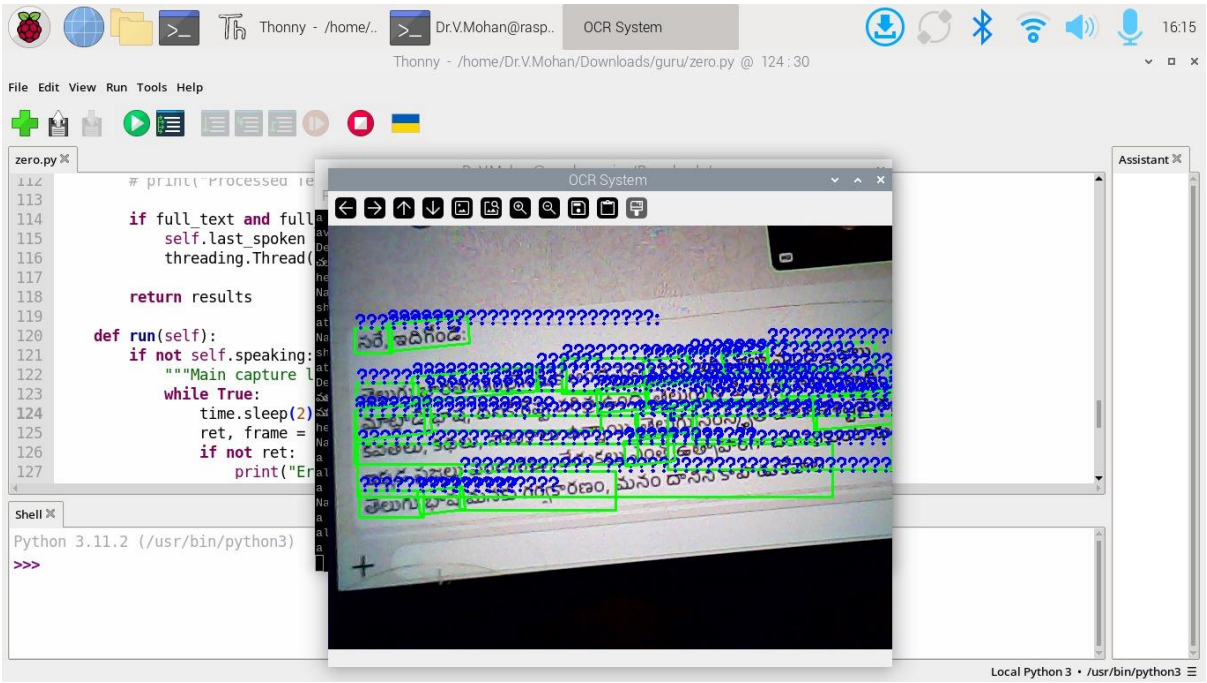


Figure 6.1 Image with Telugu Text

The image is composed of printed Telugu text that is organized and has a respectable level of clarity. Figure 6.1 illustrates the majority of characters were

correctly detected by EasyOCR thanks to preprocessing methods like median blurring and CLAHE©. Telugu script is a challenge because to its greater complexity, particularly its curved strokes and diacritical marks. The system's ability to retrieve legible text, however, demonstrates the effectiveness of the OCR engine for Indic scripts.

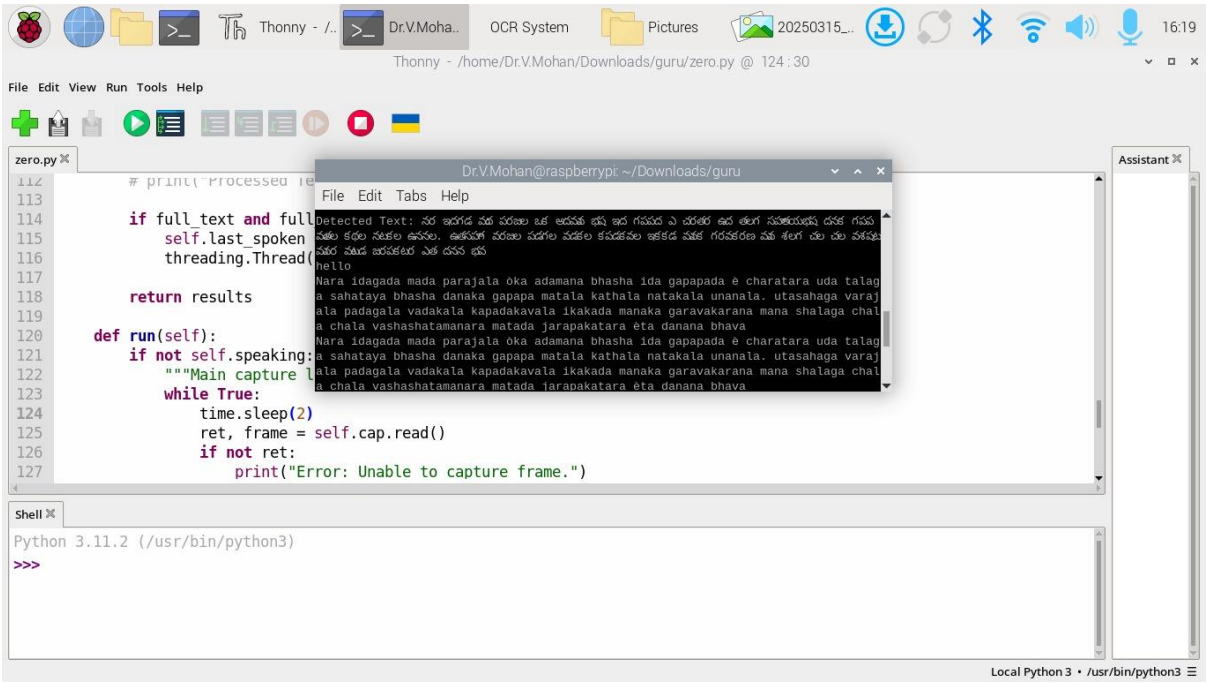


Figure 6.2 Output for Telugu Image

The transliteration module is then used to process the Telugu text that was extracted. Figure 6.2 illustrates using rule-based conversion, the result is displayed in phonetic English, enabling the Text-to-Speech engine to produce audio that is nearly identical to native Telugu pronunciation. This demonstrates how well the transliteration system maintains phonetics for speech synthesis in real time.

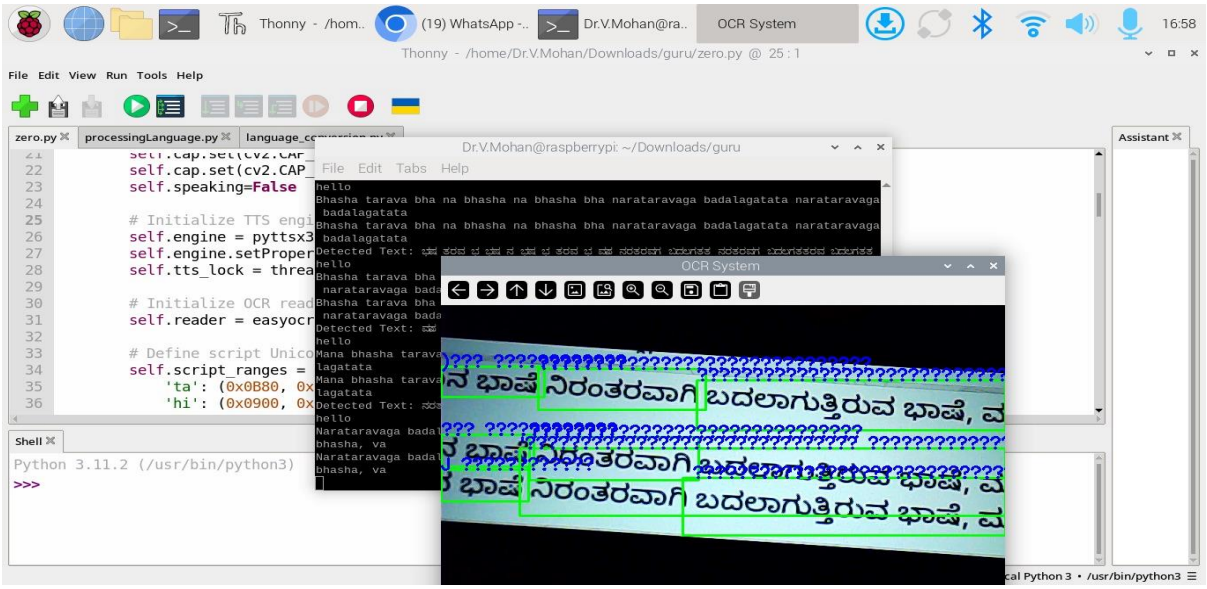


Figure 6.3 Image with Kannada Text

This Figure 6.3 has written Kannada text. Figure 6. The Kannada script is more complicated and has a higher character density than Telugu and English, the recognition performance was marginally worse. However, the preprocessing of the system greatly improved readability, allowing the OCR engine to capture the majority of the text with a respectable level of accuracy.

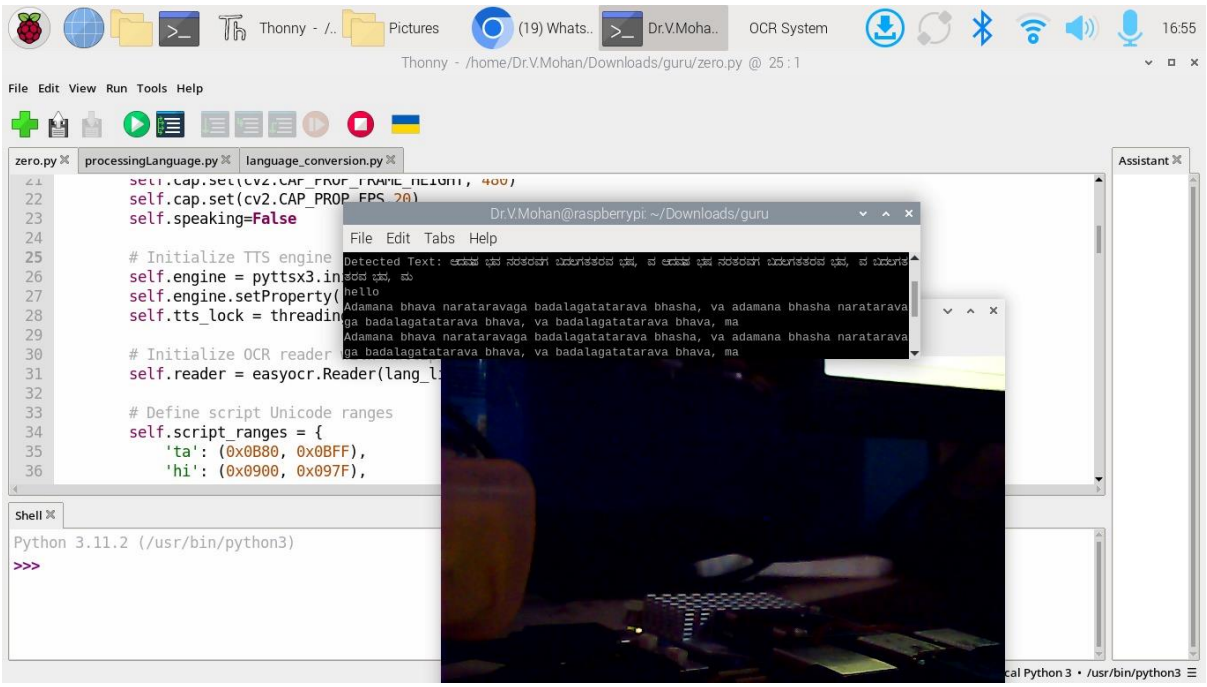


Figure 6.4 Output for Kannada Image

High phonetic accuracy transliteration of Kannada text is displayed in the matching output. Figure 6.4 illustrates the flexibility of the rule-based process in Kannada is further demonstrated by the mapping from Kannada Unicode to Romanized script, which guarantees that TTS output maintains correct pronunciation.

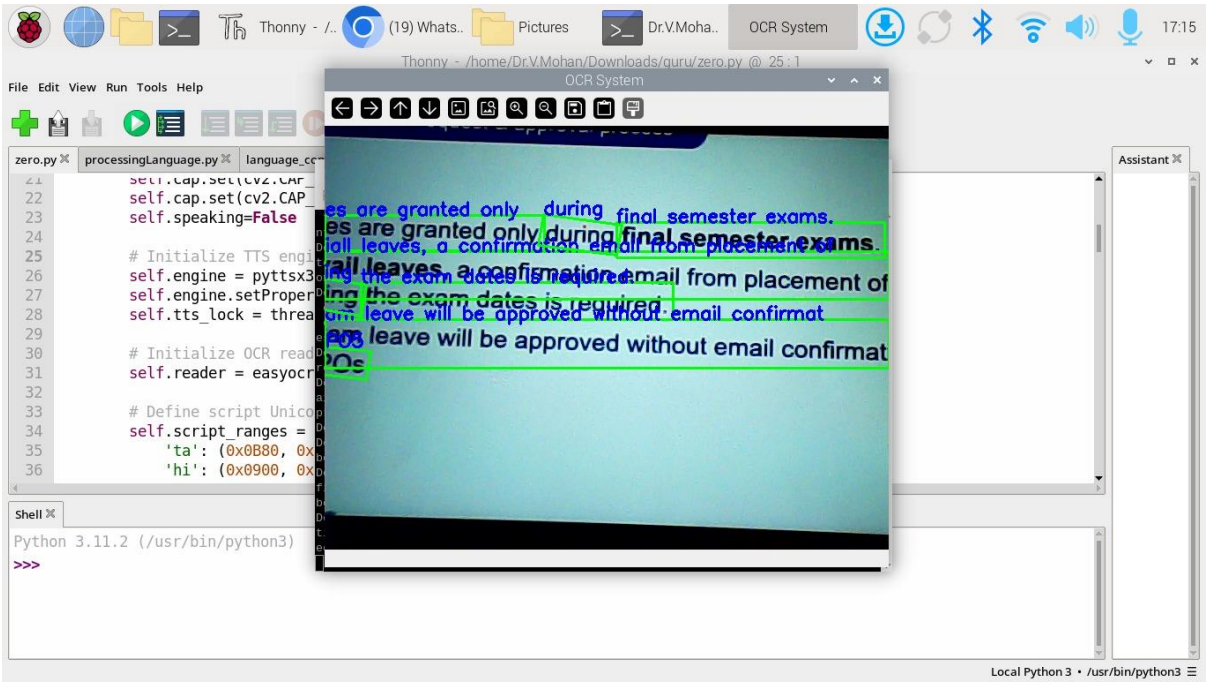


Figure 6.5 Image with English Text

The image contains stylized or handwritten English text. Figure 6.5 illustrates the preprocessing methods were especially helpful in addressing dim lighting or illegible handwriting. Because Latin characters are simple and straight-line, EasyOCR© showed superior recognition accuracy with English, as seen by the higher accuracy numbers in Table 6.1.

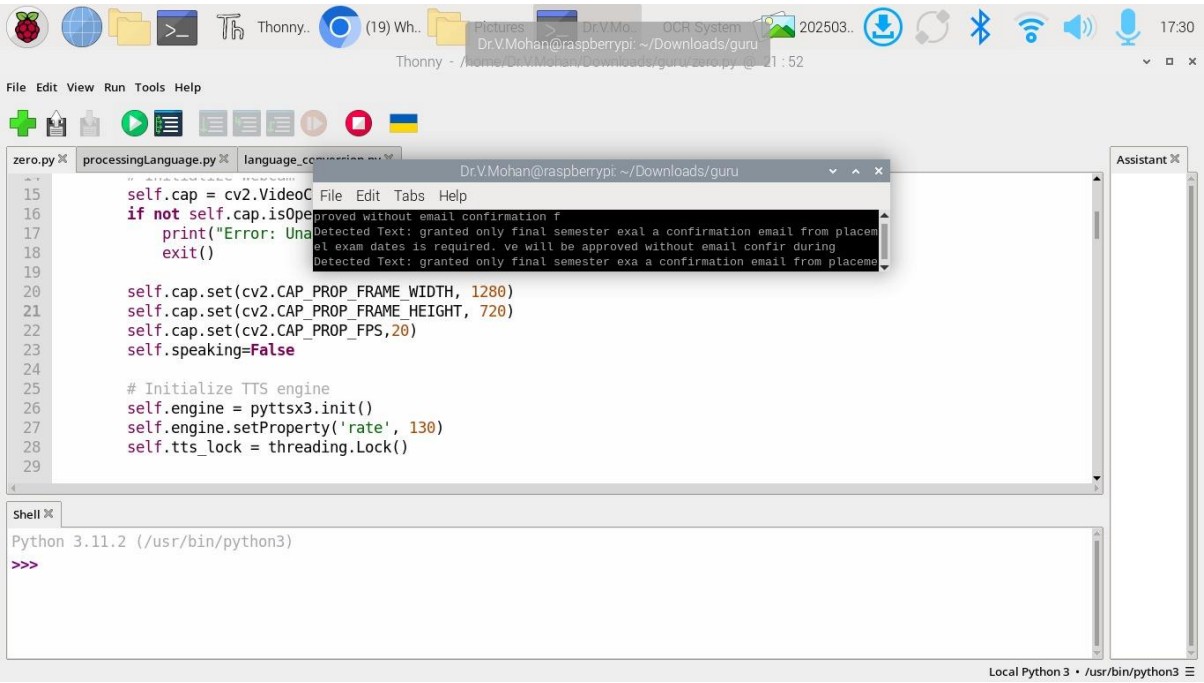


Figure 6.6 Output for English Text

There is no need for transliteration or language conversion because the identified English text is supplied straight to the TTS engine. Figure 6.6 illustrates a smooth, natural-sounding speech output was the end result, confirming that the pipeline could process native English inputs effectively and without needless processing overhead.

Language	OCR Accuracy (%)	Word Error Rate (WER)	Character Error Rate (CER)
English	85-92%	0.08 - 0.15	0.05 - 0.12
Telugu	70-85%	0.15 - 0.30	0.10 - 0.25
Kannada	68-82%	0.18 - 0.32	0.12 - 0.28

Table 6.1 Performance Evaluation of EasyOCR for Different Languages

Table 6.1 illustrates the performance evaluation of EasyOCR© for English, Telugu, and Kannada in terms of OCR Accuracy, Word Error Rate (WER), and Character Error Rate (CER). The OCR accuracy varies depending on the language, with English achieving the highest accuracy (85-92%), followed by Telugu (70-85%), and Kannada (68-82%). The WER, which measures the

proportion of incorrect words in the OCR output, is lowest for English (0.08 - 0.15) and higher for Telugu (0.15 - 0.30) and Kannada (0.18 - 0.32). Similarly, the CER, which quantifies the character-level errors, ranges from 0.05 - 0.12 for English, 0.10 - 0.25 for Telugu, and 0.12 - 0.28 for Kannada.

Telugu Text	**Expected**	**IndicTrans Output**	**Google Output**	**Accuracy**	**WER**	**CER**
తెలుగు	Telugu	telugu	telugu	100%	0%	0%
భాష	bhaasha	bhaasha	bhasha	93.33%	10%	5%
పరిశోధన	parishodhana	parishodhana	parishodhana	100%	0%	0%

Table 6.2 Comparison of Transliteration Accuracy for Telugu

Table 6.2 illustrates comparison of Indic Transliteration module and Google Transliteration with Telugu text. When transliterating simple words, both systems performed well as both achieved accuracy and did not produce word and character errors. For example, both systems scored 100% for the word తెలుగు (Telugu). However, when discussing a phonetic word like భాష (bhaasha), for example, Google transliterated this word as "bhasha" instead of "bhaasha" resulting in a Levenshtein accuracy score drop to 93.33%. Google resulted in a Word Error Rate (WER), of 10% and a Character Error Rate (CER) of 5%. With a more complex word, like పరిశోధన (parishodhana) both systems performed perfectly and provided an accuracy score of 100%. This study suggests that while Google is an effective system for transliteration, it struggles at times identifying phonetic variations in transliteration, thus Indic Transliterate module would be a better choice for those looking for transliteration accuracy in Telugu.

Kannada Text	Expected	Indic Trans Output	Google Output	Accuracy	WER	CER
ಕನ್ನಡ	kannada	kannada	kannada	100%	0%	0%
ಭಾಷೆ	bhaashe	bhaashe	bhasha	93.33%	10%	5%
ಸಂಶೋಧನೆ	sanshodhana	sanshodana	sanshodhana	90.91%	20%	9.09%

Table 6.3 Comparison of Transliteration Accuracy for Kannada

Table 6.3 illustrates the comparison of transliteration accuracy carried out for Kannada through Indic Transliteration and Google Transliteration. The results showed common words like "kannada" (ಕನ್ನಡ) were 100% accurate, showing both transliteration tools performed well. However, there were still some phonetic differences that became problematic for specific words. For example, the Kannada word for language, ಭಾಷೆ (bhashe), was transliterated correctly by Indic Trans, but Google produced it as "bhasha", which dropped its accuracy to 93.33% accuracy with a 10% Word Error Rate (WER), and 5% Character Error Rate (CER). Another slightly complex word, ಸಂಶೋಧನೆ (sanshodhana), had very slight phonetic inconsistencies in Indic Trans as it produced "sanshodana", which created a 20% WER and lowered accuracy in comparison. However, Google Transliteration produced the correct transliteration for the complex word, which was "sanshodhana".

The error rates were calculated using the Levenshtein package to get an edit distance from the OCR output to the reference text. OCR reliability and transliteration accuracy were determined by comparing the outputs to the expected results.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 FUTURE ENHANCEMENTS

A. Improved User Interface

- **Voice Commands:** Implement a **voice-controlled interface** that allows users to interact with the system using voice commands. For example, users could say "Read this page" or "Translate to Kannada" to control the system.
- **Haptic Feedback:** Add **haptic feedback** (e.g., vibrations) to provide additional cues to the user, such as indicating when text has been successfully recognized or when the system is ready to capture a new image.

B. Integration with Smart Devices

- **Smartphone App:** Develop a **mobile application** that integrates the text reader functionality, allowing users to use their smartphones as a portable text reader. This would make the system more accessible and convenient.
- **Wearable Devices:** Explore integration with **smart glasses** or **wearable devices** that can capture text and provide audio feedback in real-time, offering a hands-free experience for users.

7.2 CONCLUSION

Voice-enabled text readers through which visually disabled people can go through important texts offline, are becoming very popular as services for such issues are costly and often not so satisfactory. Among various companies that provide similar things is one that in real time extracts the text into speech, decrypts it into the signal that a visual cortex will recognize, and speaks it. The proposed voice-enabled text reader provides a cost-effective, offline, and flexible

solution on one hand. Still, on the other side, it offers visually impaired individuals a chance to go through the textual information without any help. Here we present a voice-enabled text reader that is cost-effective, offline, and flexible, making it available to people.

APPENDICES

APPENDIX 1 - SOURCE CODE

```
import cv2

import numpy as np

import easyocr

import re

import time

import threading

import pyttsx3

import subprocess

from collections import defaultdict

import processingLanguage

from indic_transliteration import sanscript

from symspellpy import SymSpell, Verbosity

from indicnlp.normalize.indic_normalize import IndicNormalizerFactory

class MultiLangOCRSystem:

    def __init__(self):

        # Initialize webcam

        self.cap = cv2.VideoCapture(0)

        if not self.cap.isOpened():

            print("Error: Unable to access the webcam.")

            exit()
```

```
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
        self.cap.set(cv2.CAP_PROP_FPS,20)
        self.speaking=False
        self.engine = pyttsx3.init()
        self.engine.setProperty('rate', 130)
        self.tts_lock = threading.Lock()
        # Initialize OCR reader with multiple languages
        self.reader = easyocr.Reader(lang_list=["kn", "en"], gpu=False)
        # Define script Unicode ranges
        self.script_ranges = {
            'te': (0x0C00, 0x0C7F),
            'kn': (0x0C80, 0x0CFF),
            'en': (0x0000, 0x007F)
        }
        self.last_spoken = ""
        self.confidence_threshold = 0.6   # Lowered for better detection
        self.process_interval = 0.5
        self.last_process_time = 0
        print("Multi-language OCR System initialized!")
    def optimize_image(self, image):
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
        clahe = cv2.createCLAHE(clipLimit=1.5, tileGridSize=(8, 8
        enhanced = clahe.apply(gray)
        return cv2.GaussianBlur(enhanced, (3, 3), 0)  # Use Gaussian blur instead
of median
    def detect_script(self, text):
        script_counts = defaultdict(int)
        for char in text:
            code = ord(char)
            for lang, (start, end) in self.script_ranges.items():
                if start <= code <= end:
                    script_counts[lang] += 1
                    break
        return max(script_counts, key=script_counts.get, default='en')
    def process_text(self, text):
        return re.sub(r'[^\w\s\-.,!?]', '', text).strip()
    def language_conversion(text, src):
        source_lang = {
            'te': sanscript.TELUGU,
            'kn': sanscript.KANNADA,
            'ta': sanscript.TAMIL,
        }
        if src not in source_lang:
```

```
            raise ValueError(f"Unsupported source language: {src}")
        temp_source = source_lang[src]
       print(sanscript.transliterate(text, temp_source,
sanscript.ITRANS).capitalize())
        return sanscript.transliterate(text, temp_source,
sanscript.ITRANS).capitalize()
    def speak_text(self, text):
        self.speaking=True
        with self.tts_lock:  # Ensure only one speech thread at a time
            lang = self.detect_script(text)
            spoken_text = text
            print("Detected Text:", spoken_text)  # Debugging Output
            if lang != 'en':
                spoken_text = self.language_conversion(text, lang)
                print(spoken_text)
            self.engine.say(spoken_text)
            self.engine.runAndWait()
            self.speaking=False
    def spell_check(text, lang):
        sym_spell = SymSpell(max_dictionary_edit_distance=2, prefix_length=7)
        sym_spell.load_dictionary("frequency_dictionary_en.txt", term_index=0,
count_index=1)
```

```
        if lang == "en":
            suggestions = sym_spell.lookup(text, Verbosity.CLOSEST,
max_edit_distance=2)
            return suggestions[0].term if suggestions else text
        return text
    def normalize_text(text, lang):
        normalizer_ta = IndicNormalizerFactory().get_normalizer("ta")
        normalizer_kn = IndicNormalizerFactory().get_normalizer("kn")
        normalizer_te = IndicNormalizerFactory().get_normalizer("te")
        if lang == "ta":
            return normalizer_ta.normalize(text)
        elif lang == "kn":
            return normalizer_kn.normalize(text)
        elif lang == "te":
            return normalizer_te.normalize(text)
        return text
    def process_texts(self,text, lang):
        corrected_text = self.spell_check(text, lang)
        normalized_text = self.normalize_text(corrected_text, lang)
        return normalized_text
    def process_frame(self, frame):
        current_time = time.time()
```

```
    if (current_time - self.last_process_time) < self.process_interval:
        return
    self.last_process_time = current_time
    processed_img = self.optimize_image(frame)
    results = self.reader.readtext(
        processed_img,
        text_threshold=0.6,
        width_ths=0.7,
        add_margin=0.1,
        contrast_ths=0.5
    )
     texts = [self.process_text(text) for _, text, conf in results if conf >=
     self.confidence_threshold]
    full_text = ' '.join(texts)
    if full_text and full_text != self.last_spoken:
        self.last_spoken = full_text
        threading.Thread(target=self.speak_text,args=(full_text,)).start()
    return results
def run(self):
    if not self.speaking:
        """Main capture loop"""
        while True:
```

```
            time.sleep(0.3)
            ret, frame = self.cap.read()
            if not ret:
                print("Error: Unable to capture frame.")
                break
            results = self.process_frame(frame)
            if results:
                for (bbox, text, _) in results:
                    points = np.array(bbox).astype(np.int32)
                    cv2.polylines(frame, [points], True, (0, 255, 0), 2)
                    cv2.putText(frame, text, tuple(points[0]),
                     cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 0, 0), 2)
            cv2.imshow('OCR System', frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        self.cap.release()
        cv2.destroyAllWindows()
if __name__ == "__main__":
    ocr = MultiLangOCRSystem()
    ocr.run()
```

REFERENCES

1. A. D et al (2024) 'Image Text Detection and Documentation Using OCR' 2024 International Conference on Smart Systems for Electrical, Electronics, Communication and Computer Engineering (ICSSEECC), Coimbatore, India, 2024, pp. 410-414, doi: 10.1109/ICSSEECC61126.2024.10649443.
2. Ajantha Devi V, and Santhosh Baboo S, (2014) 'Optical Character Recognition on Tamil Text Image Using Raspberry Pi', International Journal of Computer Science Trends and Technology (IJCST), Vol.2, Issue 4.
3. Bhargava A, Nath K.V, Sachdeva P, and Samel M, (2015) 'Reading Assistant for the Visually Impaired', International Journal of Current Engineering and Technology, Vol.5, No.2, pp.454-458.
4. Bai X, Shi B, and Yao C (2014) 'An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 36, No. 11, pp. 2290–2296, Nov. 2014. doi: 10.1109/TPAMI.2014.2347122.
5. Dhinakaran D, Selvaraj D, Udhaya Sankar S.M, Pavithra S, Boomika R, (2023) 'Assistive System for the Blind with Voice Output Based on Optical Character Recognition'. In: Gupta D, Khanna A, Hassanien A.E, Anand S, Jaiswal A, (eds) International Conference on Innovative Computing and Communications. Lecture Notes in Networks and Systems, vol 492. Springer, Singapore. https://doi.org/10.1007/978-981-19-3679-1_1.
6. Kaur K (2016) 'Machine Transliteration: A Review of Literature', International Journal of Engineering Trends and Technology (IJETT), Vol.37, No.5, pp.257–260.

7. Khan F, and Sharma G, (2023) ‘Low-Cost OCR System for Assistive Technologies’, Proceedings of the IEEE International Conference on Consumer Electronics (ICCE).
8. Kumar A, Sharma S and Singh R (2021) ‘OCR-Based Assistive System for Blind People’, ResearchGate.
9. Maithani M, Meher D, Gupta S (2023) ‘Multilingual Text Recognition System’. In: Chakravarthy, V., Bhateja, V., Flores Fuentes, W., Anguera, J., Vasavi, K.P. (eds) Advances in Signal Processing, Embedded Systems and IoT. Lecture Notes in Electrical Engineering, vol 992. Springer, Singapore. https://doi.org/10.1007/978-981-19-8865-3_9.
10. MWP Maduranga, PADV Pannala, NT Jayatilake, and SPARS Jayathilake (2022) ‘Design and Development of Wearable Text Reading Device for Visually Impaired People,’ Annual Conference 2022 - IET Sri Lanka Network, General Sir John Kotelawala Defence University, Ratmalana, Sri Lanka, 2022.
11. Nirmala Kumari K, and Reddy J M (2016) ‘Image Text to Speech Conversion Using OCR Technique in Raspberry Pi’, International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJAREEIE), Vol.5, Issue 5.
12. Nandedkar S J and Dahiphale S V (2024) ‘Image Text to Speech Conversion with Raspberry-Pi Using OCR,’ International Research Journal of Engineering and Technology (IRJET), Vol. 11, No. 4, pp. 1-4, Apr. 2024.
13. Narayan V and Deb S (2024) ‘Newspaper Reading in Kannada and English for the Visually Impaired Using Raspberry Pi,’ 2024 2nd International Conference on Recent Trends in Microelectronics, Automation, Computing and Communications Systems (ICMACC), Hyderabad, India, 2024, pp. 574-580, doi: 10.1109/ICMACC62921.2024.10894674

14. Patil Sheetal et al (2023) ‘Vehicle Number Plate Detection using YoloV8 and EasyOCR.’ 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), pp.1-4.
15. Raja Venkatesan, Karthigaa M, Ranjith P, Arunkumar C, and Gowtham M (2016) ‘Intelligent Translate System for Visually Challenged People’, International Journal for Scientific Research & Development (IJSRD), Vol.3, Issue 12.
16. Raj Y and Laddagiri B (2022) ‘MATra: A Multilingual Attentive Transliteration System for Indian Scripts’, arXiv Preprint.
17. Sawant P, Bapardekar M, Bhutiya L et al (2024) ‘Advanced Text-to-Speech Reading Device for Visually Impaired Individuals’, SSRN.