Journal of Machine Engineering, 2025, Vol. 25
ISSN 1895-7595 (Print) ISSN 2391-8071 (Online)

Received*: 20 April 2025* / Accepted: *15 August 2025* / Published online: *21 August 2025*

optical character recognition,
assistive technology,
braille translation,
real time data processing

Yogita DUBEY[1*], Vijay WATH[1],
Yadhnesh VYAWAHARE[1], Vaishnavi DEOGADE[1],
Gagan LOYA[1], Akash PRATAPE[1],
Darshana CHHATANI[1]

AN INTEGRATED OCR-BASED ASSISTIVE SYSTEM FOR VISUALLY IMPAIRED INDIVIDUALS WITH ENHANCED ACCESSIBILITY

Mainstream technologies for assisting specially-abled individuals have evolved in both printed and digital mediums. This research paper presents a study on a system designed to assist specially-abled individuals using Optical Character Recognition (OCR). It also explores modern-day solutions for real-time data accessibility. The OCR system is integrated with a wide range of hardware to enhance accessibility and convenience. The hardware includes keyboards, displays, buzzers, controllers, actuators, and more. For real-time data access, a web server is provided for manual data input, which is then processed and recognized using the software. The input data can be digital, manually typed, or in the form of various file types. Additionally, a webcam is set up to capture and process data from the surroundings for recognition. The software extends its functionality to handwritten notes and other forms of data. It can differentiate between numerals, alphabets, and symbols. The recognized data is then translated into the required Braille format, specifically arrays corresponding to each letter. The translated data is subsequently transmitted to the hardware for appropriate feedback. This research paper also includes a comparative analysis of three widely recognized OCR models – EasyOCR, Pytesseract, and SuryaOCR. The analysis evaluates various performance aspects, including speed, processing time, accuracy, complexity, dependencies, error rate, and error-handling capacity.

1. INTRODUCTION

Assistive technology has revolutionized accessibility for individuals with visual impairments, enabling them to interact with printed and digital materials more effectively. Assistive technology has significantly transformed accessibility for individuals with visual impairments, enabling them to interact with printed and digital materials in various ways [1]. The system is designed to process text inputs from multiple sources, including uploaded documents, typed text, and captured images. OCR technology is employed to extract, process,

[1] Department of Electronics and Telecommunication Engineering, Yeshwantaro Chavan College of Engineering, Nagpur, India
[*] E-mail: yogeetakdubey@yahoo.co.in
https://doi.org/10.36897/jme/209567

and convert both handwritten and printed text into a physical Braille format for individuals who are blind or visually impaired. Over the years, several OCR models have been proposed and refined, including lexicon-free deep learning-based OCR techniques [2] and sequence-to-sequence learning approaches [3]. An event-driven control system ensures smooth and efficient user interactions, such as menu navigation, text input, and selection confirmation. A buzzer, vibrations from the Braille module, and visual alerts on the OLED screen provide real-time feedback to users.

A key aspect of this study is the evaluation of three OCR models, EasyOCR, Pytesseract, and SuryaOCR, across different scenarios. The assessment offers a comprehensive comparative analysis, focusing on their recognition of printed and handwritten text across various font styles, sizes, orientations, and noise levels. Previous studies have explored efficient and scalable text recognition frameworks [4], transformer-based OCR methods [5], and post-processing pipelines leveraging natural language processing to enhance OCR accuracy [6].

Factors considered in the evaluation include the models' ability to handle complex distortions and low-quality images, as well as their accuracy in text recognition and multilingual support. To ensure a robust evaluation, various OCR benchmarking tools and methodologies have been reviewed. Character-level and word-level evaluation frameworks have been proposed to systematically assess OCR performance [7]. Previous research has also examined automatic end-to-end evaluation for scene text OCR [8]. Additionally, open-source OCR evaluation tools [9] and comprehensive surveys on OCR evaluation metrics provide further insights into the effectiveness of different recognition approaches [10].

A comprehensive survey on emerging assistive technologies is presented in [11], [12]. Recently, the integration of Internet of Things (IoT) technologies has led to the development of refreshable OCR-Braille solutions, enabling real-time text detection and Braille translation [13]. The deep learning-based approach named Fly-LeNet is proposed in [14] that converts Braille images into multilingual texts, enhancing accessibility for visually impaired individuals. This study focuses on developing and evaluating a unified assistive system that integrates optical character recognition (OCR), Braille translation, and wireless communication to provide immediate text accessibility. By combining hardware components such as an OLED display, a camera module, tactile buttons, a buzzer, and a keyboard interface, the system ensures a user-friendly and interactive experience. Additionally, a web-based wireless server allows users to submit text remotely, expanding accessibility beyond physical interactions.

2. METHOD

The methodology employed for assistive system for visually impaired person is shown in Fig. 1. When the system is started, it runs a boot sequence that sets up various libraries, and all the GPIO peripherals are initialized. An LED indicator connected to GPIO blinks when the boot sequence is running and turns off when the boot sequence is completed, indicating that the system is ready. For image processing and hardware communication time, board, busio, and PIL are used. We have used the Adafruit SSD 1306 128×64 pixels OLED display, which is I2C-based. Additionally, there are 3 functions for user inputs and wireless Communication. There are 5 buttons for up, down, ok, back, and next. Each button has a specific debounce time to remove noise from signals. A buzzer is also connected to give auditory responses when buttons are pressed to increase intuitive interaction. The main interface of the system is shown by the OLED display. The script dynamically creates menu items and moves the cursor based on button presses. The menu contains 3 options. Camera, Wireless, and Keyboard. The system has a retry mechanism that tries to connect to the peripheral with 1 second to tackle potential errors in I2C communication. This approach to error handling guarantees dependable and consistent display functionality. The system enters a non-blocking infinite loop that pauses for 100 milliseconds in each iteration. This mechanism ensures system responsiveness while reducing CPU usage. This combination of feedback mechanism makes it suitable for research as well as practical applications. The custom tactile input modules used for user interaction and feedback are shown in Fig. 2. Figure 2(a) and 2(b) shows the braille modules in different angles and Fig. 2(c) shows the conversion of text. These modules were designed for intuitive operation and tested for integration with the complete system.

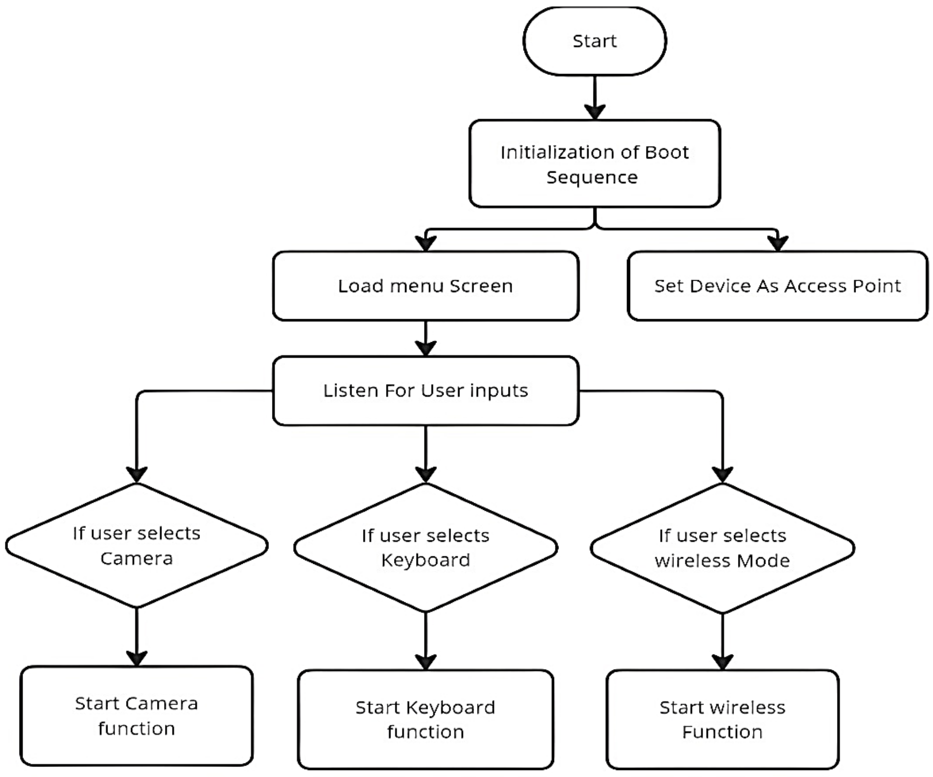


Fig. 1. Basic Flow Diagram

The camera functionality is executed as flow shown in Fig. 3. As the user selects the camera option, the Picamera2 library for capturing images and the gpiozero library for GPIO management are initialized. Also, the indicator LED is turned on, and 'Click Picture' is shown on the OLED screen to indicate the initialization of the camera sensor. To take a picture, the user needs to press OK. This triggers a utility function that takes and saves the image in a specific file path. After taking an image, the OLED screen displays 'Picture taken' for better user understandability. After taking an image, the system extracts the text from the image. While this is happening, the OLED shows 'Loading,' and the indicator LED starts blinking to indicate the ongoing process. The OCR (Optical Character Recognition) process is carried out by invoking the OCR Model. The validity of the extracted text is then checked. If no text is detected, the OLED screen shows 'No text found' to notify the user. Otherwise, the extracted text is sent to the decoder assembly controller linked to the Raspberry Pi through the I2C bus. The custom braille modules activate the Braille display, enabling users to read the entered text through touch Throughout the entire process, the system stays responsive to user interaction. Pressing the button_back at any time stops the camera operation and brings the user back to the main menu. The Picamera2 module is appropriately halted once the process wraps up, ensuring effective resource management.

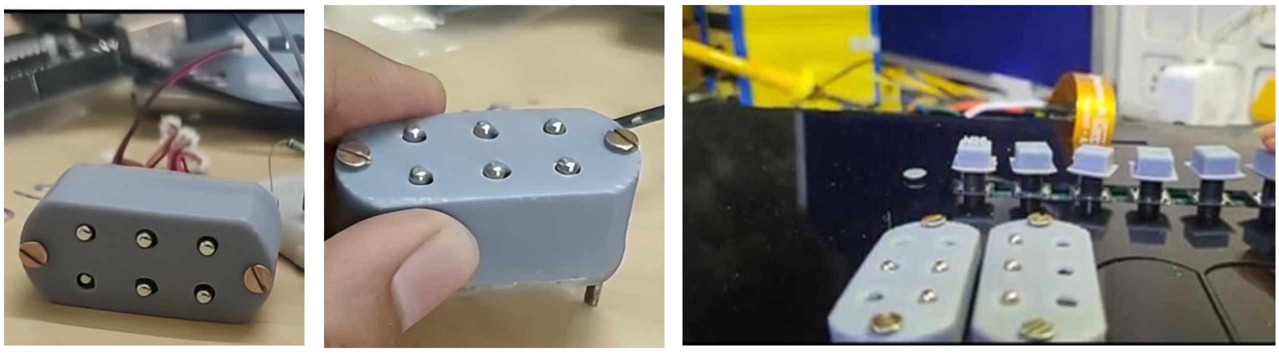

(a) (b) (c)

Fig. 2. Braille Modules and conversion of text

The functionality of the keyboard is shown in Fig. 4. The system utilizes the evdev library to capture keystroke events from an external USB keyboard. The procedure starts by scanning all existing input devices. This involves iterating through device paths, identifying devices with names that suggest they are keyboards (like those containing “kbd” or “keyboard” keywords), and confirming that the device is capable of key events (EV_KEY). Once a keyboard is successfully identified, the device is returned for further engagement. Once a keyboard is identified, the OLED display indicates “Keyboard Ready” while the indicator LED lights up, signalling that the system is set to accept text input. Users see the prompt “Type below” on the display.

As the user types, every keystroke event is captured and processed. The system manages typical text input functions: alphanumeric characters are added to the input string, pressing the spacebar inserts a space, and pressing backspace removes the last character if there is one. The OLED display updates in real-time to show the user's current input. When the Enter key is pressed, the system displays “LOADING...” as the entered text is sent to the decoder assembly controller connected to the Raspberry Pi via the I2C bus. The custom braille modules then activate the Braille display, allowing users to read the entered text by touch.

The system incorporates features for user control and error management. If the button back is pressed during the input phase, the keyboard reading operation halts, and the system reverts to the main menu. Furthermore, if no keyboard is found, the OLED screen shows a

"Connect Keyboard" message to assist the user in troubleshooting the hardware connection. The design prioritizes accessibility and user-friendliness for individuals with visual impairments. The combination of real-time OLED notifications, responsive keyboard input, and tactile Braille output creates a smooth interface for text entry and Braille translation.

The functionality of the wireless server is depicted in Fig. 5. It is developed using Flask, offering a web-based interface for text submission, PDF handling, and image-to-text translation, integrated with a Braille display system. The server operates on a Raspberry Pi with a specific IP address, enabling clients to connect wirelessly and provide text, PDF documents, or images for conversion to Braille. Initially, the application sets up an upload directory and configures the OLED display to show a welcome message. It then establishes routes to manage different requests, offering immediate feedback to users via the OLED display. The server supports several key functionalities given below.

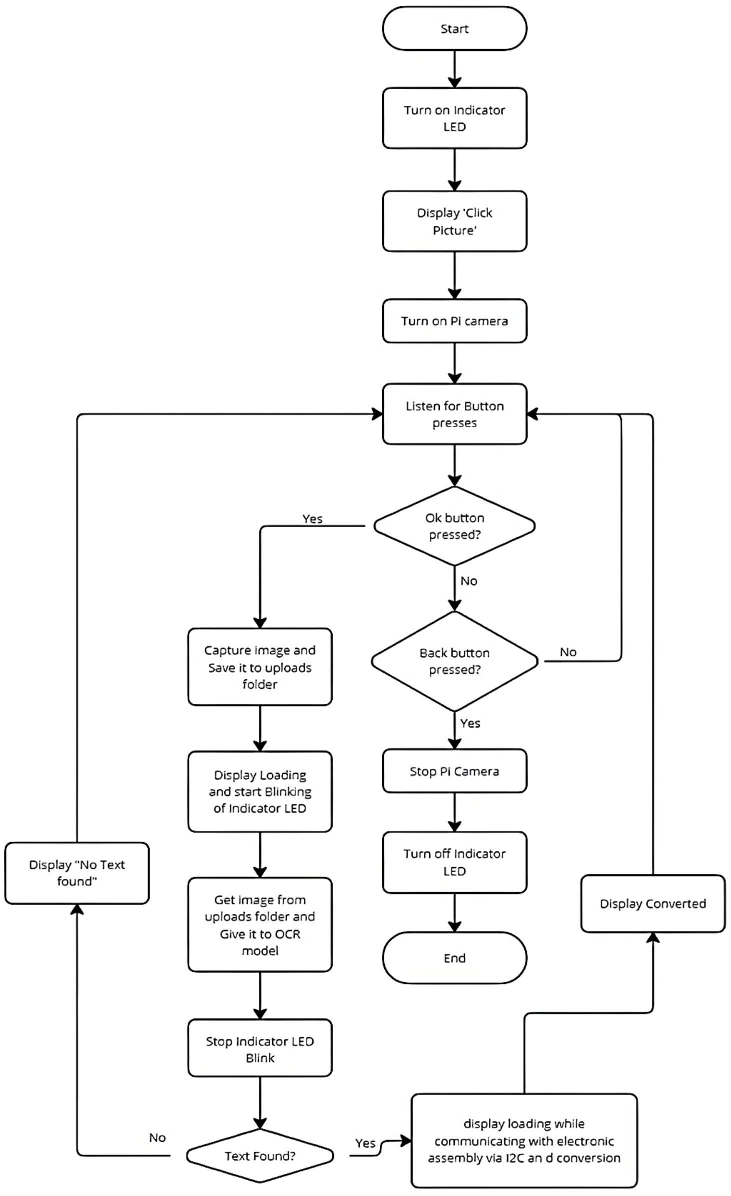

Start
Turn on Indicator LED
Display 'Click Picture'
Turn on Pi camera
Listen for Button presses
Ok button pressed?
Yes
No
Capture image and Save it to uploads folder
Back button pressed?
No
Yes
Display Loading and start Blinking of Indicator LED
Stop Pi Camera
Turn off Indicator LED
Display "No Text found"
Display Converted
Get image from uploads folder and Give it to OCR model
End
Stop Indicator LED Blink
No
Text Found?
Yes
display loading while communicating with electronic assembly via I2C an d conversion

Fig. 3. Camera Flow Diagram

- Text Processing: Users can submit text through a POST request to the /process_text endpoint. The OLED screen shows the entered text, which is then transmitted to the decoder assembly controller connected to the Raspberry Pi via the I2C bus.

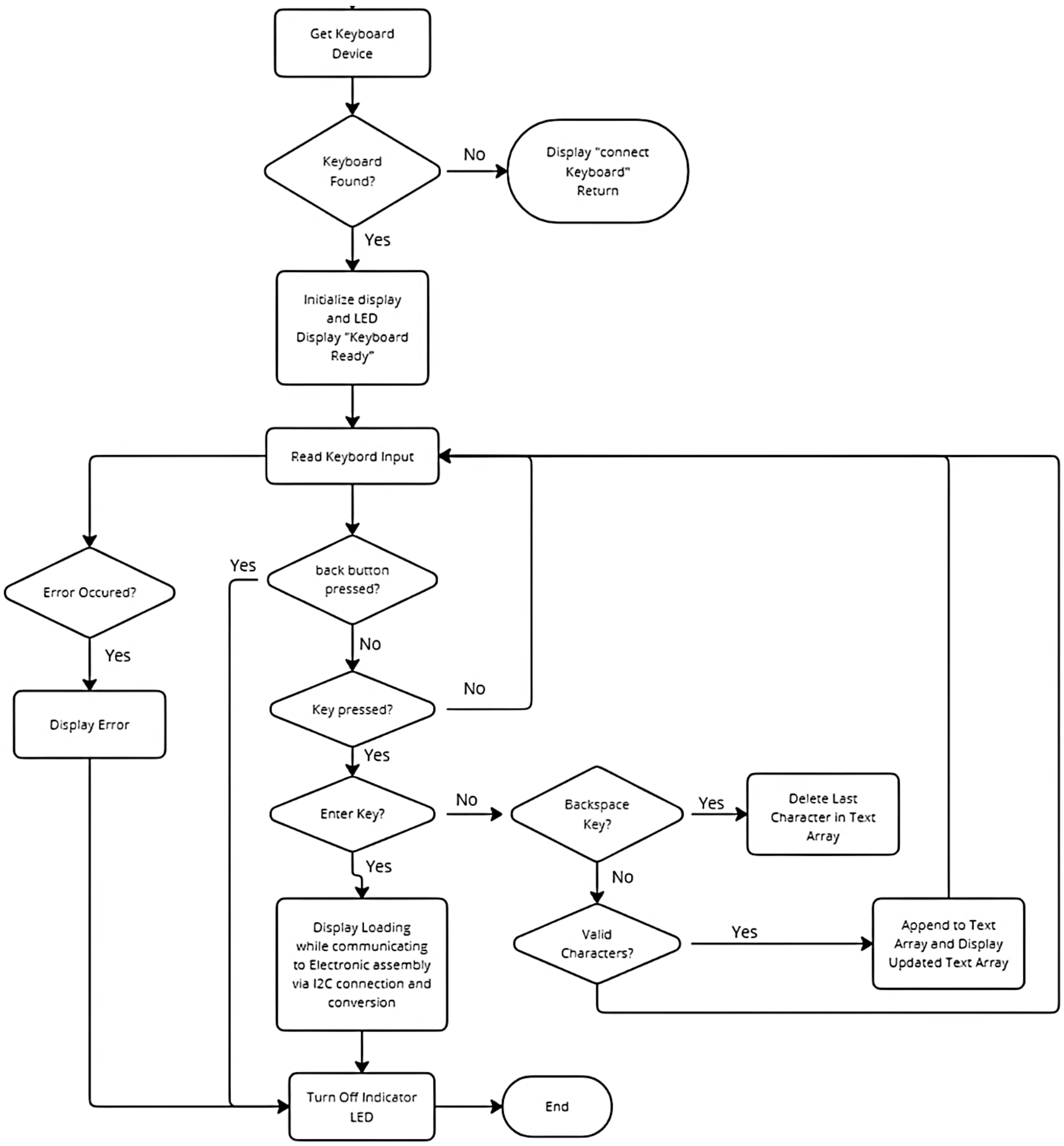

Get Keyboard Device
Keyboard Found?
No
Display "connect Keyboard" Return
Yes
Initialize display and LED Display "Keyboard Ready"
Read Keybord Input
Error Occured?
Yes
Display Error
back button pressed?
Yes
No
Key pressed?
No
Yes
Enter Key?
No
Backspace Key?
Yes
Delete Last Character in Text Array
No
Valid Characters?
Yes
Append to Text Array and Display Updated Text Array
Yes
Display Loading while communicating to Electronic assembly via I2C connection and conversion
Turn Off Indicator LED
End

Fig. 4. Keyboard Flow

- PDF Processing: Users can upload PDF documents at the /process_pdf endpoint. The server employs the fitz library (PyMuPDF) to extract text from each page. Navigation through the pages is facilitated by hardware buttons, and selected content activates custom Braille modules, allowing users to read the text by touch through the Braille display.
- Image Processing: The /process_image endpoint allows users to submit image files, where OCR is applied for text extraction, and the detected text is displayed on the OLED

screen. If text is found, it is sent to the decoder assembly controller connected to the Raspberry Pi via the I2C bus.

- Server Shutdown: The server can be shut down via the /shutdown endpoint, which halts the server completely**.**
- To evaluate the performance of three OCR models–EasyOCR, Pytesseract, and SuryaOCR–controlled experiments were conducted using diverse image datasets containing printed and handwritten text in various fonts, sizes, orientations, and noise levels. The evaluation was performed under uniform conditions to ensure a fair comparison.

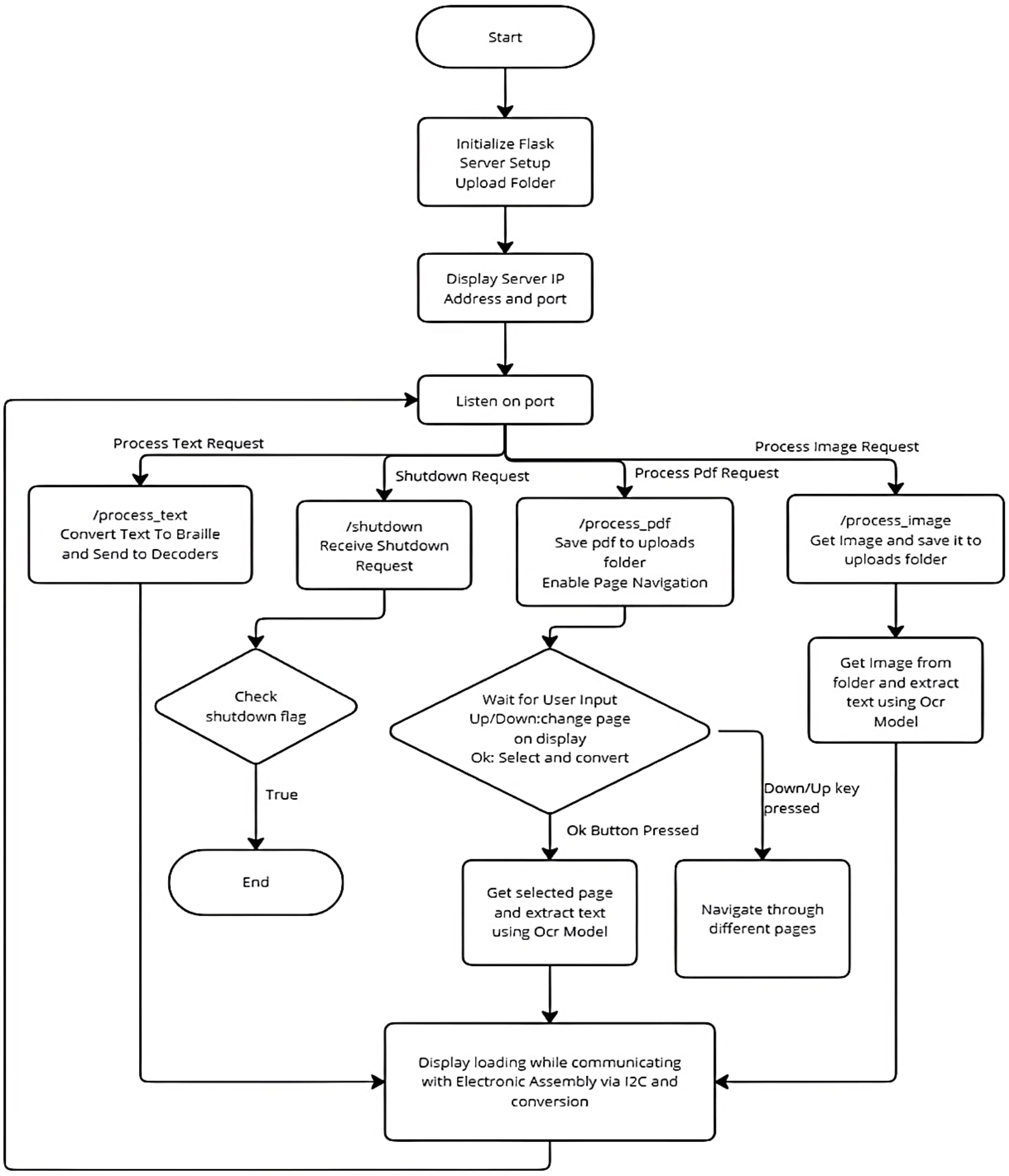

Start
Initialize Flask Server Setup Upload Folder
Display Server IP Address and port
Listen on port
Process Text Request
Shutdown Request
Process Pdf Request
Process Image Request
/process_text Convert Text To Braille and Send to Decoders
/shutdown Receive Shutdown Request
/process_pdf Save pdf to uploads folder Enable Page Navigation
/process_image Get Image and save it to uploads folder
Check shutdown flag
True
End
Wait for User Input Up/Down:change page on display Ok: Select and convert
Ok Button Pressed
Down/Up key pressed
Get Image from folder and extract text using Ocr Model
Get selected page and extract text using Ocr Model
Navigate through different pages
Display loading while communicating with Electronic Assembly via I2C and conversion

Fig. 5. Wireless Server Flow

2.1. EVALUATION CRITERIA

- Accuracy Assessment: The analysis of recognition accuracy was conducted using ground-truth datasets alongside manually validated transcriptions. Three tiers of accuracy were taken into account. Character-level accuracy focused on misrecognized, substituted, or absent characters. Word-level accuracy evaluated the successful identification of complete words, including spacing and common compound word challenges. Sentence-level accuracy examined punctuation, capitalization, grammar integrity, and logical coherence in the output. The font sizes in the test images varied from 6 pt. to 48 pt., encompassing both machine-printed and handwritten texts, using an assortment of typefaces such as serif, sans-serif, and cursive styles.
- Processing Speed: OCR models were evaluated on a dataset of 500 images that included various resolutions and content types. This comprised 200 standard A4 documents at 300 DPI, 150 low-resolution images at 72 DPI (compressed JPEGs), 100 high-resolution images at 600 DPI (TIFF and PNG formats), and 50 handwritten pages. The average processing speed (milliseconds per image) was determined for each group of resolutions. It was noted that while performance showed significant enhancement from 72 DPI to 300 DPI, the transition from 300 to 600 DPI resulted in only slight improvements. At 72 DPI, recognition accuracy declined by as much as 35%, particularly in images featuring small fonts, artifacts, or high compression levels.
- Multilingual & Script Support: The models underwent evaluation across 10 languages, which included English, French, Chinese, Japanese, Korean, Hindi, Tamil, Telugu, Bengali, and Arabic. The focus was on recognition accuracy for each script, especially in cases where ligatures, diacritics, or non-Latin character sets added complexity. The tests encompassed documents in single languages as well as mixed-language content within the same image. The decline in performance for multilingual inputs was more pronounced in scripts with intricate typography (e.g., Indic and East Asian languages), especially when font sizes fell below 10 pt.
- Noise and Distortion Handling: OCR models were exposed to a broad spectrum of visual noise and geometric distortions. Gaussian blur was applied with sigma values ranging from 1.0 to 3.5 to mimic various degrees of focus loss. Salt-and-pepper noise was introduced at 2%, 5%, and 10% pixel corruption levels, while Gaussian noise was applied with standard deviations of 5, 15, and 30 grayscale units. Compression artifacts were evaluated using JPEG images with quality levels decreased to 30%, 50%, and 70%, simulating real-world degradation seen in screenshots and low-quality scans. Geometric distortions included text rotated from ±5° to ±45°, perspective warps with corner offsets up to 20%, and curved baselines created using sine wave transformations with amplitudes between 5 and 25 pixels. Background complexity was also added using images that contained stamps, watermarks, gradients, and low-contrast text regions with contrast ratios as low as 3:1. Some examples included occlusions or overlapping graphics that obscured 20–40% of the text.
- Datasets Used: The evaluation was executed using publicly available and open-source datasets. For printed text, datasets such as ICDAR 2013, ICDAR 2015, and SynthText were utilized. Scene text recognition was assessed using IIIT-5K, SVT, and COCO-Text.

Handwriting recognition drew from the IAM Handwriting Database for English cursive writing. For multilingual and Indic scripts, datasets like MLT (Multi-Lingual Text), BanglaWriting and UNLV-ISRI were employed. These datasets encompassed a wide array of image conditions including varying font styles, scripts, resolutions, distortions, and real-world complexities to facilitate thorough evaluation across diverse scenarios.

2.2. PERFORMANCE METRICS

To quantify OCR efficiency, the following mathematical performance metrics were used.

- Character Error Rate (CER) CER evaluates character-level recognition accuracy [8] given by CER = $\frac{S+I+D}{N}$, where, S is substitutions (incorrectly recognized characters), I is insertions (extra characters added incorrectly), D is deletions (missing characters) and N is total characters in the ground truth. Lower CER values indicate higher accuracy.
- Word Error Rate (WER): WER measures errors at the word level given by WER = $\frac{S+I+D}{W}$, where, S is substitutions (incorrectly recognized words), I is insertions (extra words added incorrectly), D is deletions (missing words) and W is the total words in the ground truth. A lower WER indicates better word recognition.
- Processing Time (T): The average processing time per image was calculated as T= $\frac{T_{Total}}{N_{images}}$, where, T_{total} is the total processing time across all images, N_{images} is the number of images processed. Processing time was measured under CPU execution (single-threaded and multi-threaded runs) and GPU acceleration (for models supporting CUDA-based inference).

2.3. IMPLEMENTATION & TESTING ENVIRONMENT

In order to guarantee consistent and replicable outcomes, every OCR model was evaluated in a regulated computing setting featuring these specifications: Hardware setting of CPU (BCM2712), GPU (Broadcom VideoCore VII), RAM (8Gb LPDDR4) and storage of 128 GB sd Card. Software setting of operating system (Debian based Rasberry pi os(64Bit)), Python with OpenCV, NumPy, Tesseract-OCR, PyTorch, Easy-Ocr, Surya Git Repository

2.4. DATASET USED FOR TESTING

In order to perform a thorough assessment, the OCR tools were evaluated using a varied dataset that included printed text (various fonts and sizes), handwritten samples (cursive and block letters), mixed text documents (scanned books, invoices, business cards), multilingual text (Latin, Chinese, Arabic, Devanagari, Tamil scripts) and low-quality images (blurred, rotated, and noisy text). Every model was assessed under the same conditions to ensure a fair comparison of their performance.

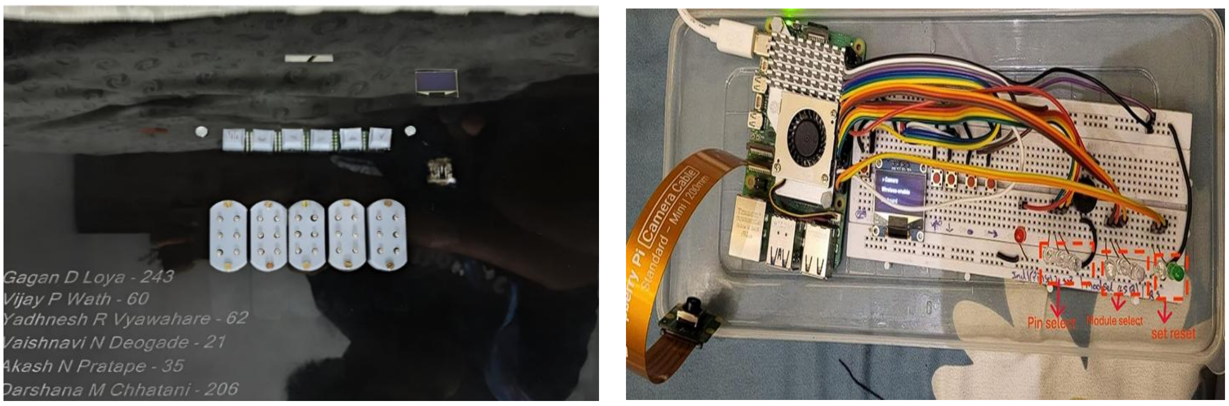
2.5. MODEL-SPECIFIC CONSIDERATIONS

Three models with following specifications were evaluated.

- EasyOCR model is based on deep learning (CNN + LSTM using PyTorch) which delivers high accuracy for printed text and moderate performance for handwritten text and offers rapid inference, particularly on CUDA-capable GPUs. But faces challenges with extreme noise and overlapping text and has limited capabilities in optimizing structured documents (tables, invoices)
- Pytesseract (Tesseract OCR) is a conventional OCR model (incorporates LSTM from Tesseract 4.0 onward) and is most effective for dealing with structured documents (invoices, forms, books). But this model has difficulty with handwriting and necessitates significant preprocessing and slower compared to EasyOCR due to a character-segmentation methodology. It is not optimized for GPU use, relying solely on CPU for processing.
- SuryaOCR is an advanced AI-based OCR system utilizing Transformer-based recognition and optimized for multilingual text, with a focus on Indic scripts. It has high capabilities in s in handling noise and recognizing handwritten text and employs deep learning techniques for denoising and correcting errors. It is compatible with CUDA and TensorRT for real-time OCR processing.

3. RESULTS AND DISCUSSION

The models underwent testing using a varied collection of documents, which comprised both printed and handwritten text across different languages, font sizes, and degrees of noise and distortion. The findings are evaluated based on criteria such as accuracy, processing speed, integration simplicity, support for multiple languages, and noise management. We have used timeIt a python library we have used timeIt python library which gets the execution time of code snippets. Fig. 6(a) shows final assembled prototype of the assistive system for visually impaired users.



(a) (b)

Fig. 6 Final Assembly with test bemch

The layout includes six custom tactile input modules, INMP441Mic, an OLED display for visual feedback and associated control circuitry mounted on a black acrylic panel. Contributor names are engraved for documentation and presentation purposes. This complete hardware integration was used during testing and demonstration phases. Figure 6(b) shows Test Bench, the setup integrates a Raspberry Pi with a camera module and a breadboard containing control buttons and module array control pins (represented by LEDs). It was used to evaluate OCR performance under diverse input conditions, aiding in real-time testing and validation of the system.

3.1. ACCURACY

The CER and WER analysis of the proposed method using SuryaOCR, EasyOCR, and Pytesseract shown in Fig. 7 shows that SuryaOCR achieved the highest accuracy, particularly excelling in complex document processing due to its transformer-based recognition and advanced deep learning denoising techniques. It exhibited exceptional performance in recognizing multilingual text and extracting handwritten content, significantly surpassing the other two models. For printed text, a CER of 1.5% and a WER of 2.3% were recorded, while for handwritten data, a CER of 5.8% and a WER of 7.5% were observed. EasyOCR performed well for printed text, especially in recognizing multiple languages, but exhibited moderate effectiveness with handwritten content. It struggled with extreme distortions and low-resolution images. For printed text, a CER of 3.2% and a WER of 5.1% were recorded, whereas for handwritten data, a CER of 14.4% and a WER of 15.7% were found. Pytesseract produced consistent results for structured text, such as invoices, tables, and scanned documents, but was less effective in noisy environments. Compared to SuryaOCR and EasyOCR, it faced challenges in contextual text extraction and required preprocessing for optimal performance. For printed text, a CER of 4.8% and a WER of 6.7% were detected, while for handwritten data, a CER of 18.2% and a WER of 21.3% were observed.

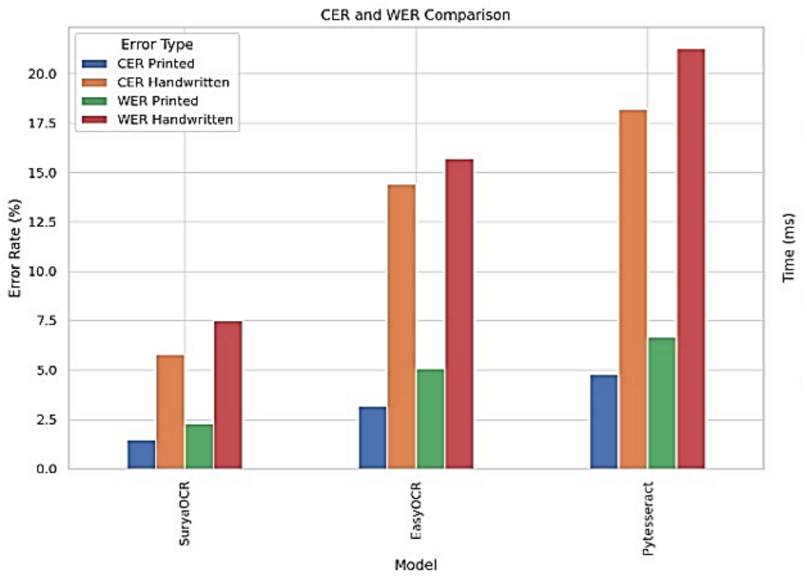


Fig. 7. Character Error Rate (CER) and Word Error Rate (WER) Analysis

3.2. PROCESSING SPEED

The comparative analysis of average processing time per frame is presented in Table 1. EasyOCR and Pytesseract demonstrated the fastest text recognition speeds, primarily due to their lightweight architectures. The utilization of CUDA enhanced EasyOCR's speed, making it well-suited for real-time applications. In contrast, SuryaOCR was slower than EasyOCR as it employed deep learning-based error correction, which reduced processing speed but significantly improved text recognition accuracy. Pytesseract had the slowest processing speed due to the absence of GPU acceleration. EasyOCR maintained higher speeds, while SuryaOCR balanced speed and accuracy, making it a strong choice for AI-enhanced OCR. The average processing time for SuryaOCR was 170 ms on CPU and 75 ms on GPU. EasyOCR was comparatively faster, with an average processing time of 140 ms on CPU and 60 ms on GPU. Pytesseract was the slowest, with an average processing time of 200 ms on CPU. The average processing time for Chowdhary et al [1] model for CPU is 160 ms while for GPU is 80 ms, For Li et al [2] model is 200 ms and 80ms respectively. In case of Namysl et al [3] and Mehta et al [6] model for CPU and GPU is 150 ms, 70 ms and 190 ms and 85 ms respectively. For handwritten text the average processing time for SuryaOCR, EasyOCR, Pytesseract, Chowdhary, Li et al, Namysl, Mehta model is 250 ms, 280 ms, 360 ms, 310 ms , 330 ms, 280 ms and 300 ms respectively. While, for high resolution image the average processing time for SuryaOCR , EasyOCR, Pytesseract, Chowdhary et al [1], Li et al [2], Namysl et al [3], and Mehta [6] model is 410 ms, 380 ms, 500 ms, 450 ms , 470 ms, 420 ms and 440 ms respectively.

Table 1. Average Processing Time per Image in milliseconds (ms)

Model	CPU (A4 Page)	GPU (A4 Page	Handwritten Text	High-Resolution Image
SuryaOCR	170	75	250	410
EasyOCR	140	60	280	380
Pytesseract	200	N/A	360	500
Chowdhary et al [1]	160	80	310	450
Li et al [2]	200	90	330	470
Namysl et al [3]	150	70	290	420
Mehta et al [6]	190	85	300	440

3.3. INTEGRATION SIMPLICITY

The evaluation of ease of use is summarized in Table 2. Pytesseract and EasyOCR received the highest scores for ease of installation due to their lightweight characteristics and low dependency needs, making them well-suited for rapid integration into Python environments. While Pytesseract functions as a wrapper for Tesseract-OCR and necessitates additional pre-processing steps to improve accuracy, such as noise reduction and thresholding, it does slightly increase the development effort involved. On the other hand, EasyOCR is delivered with pre-trained language models and requires minimal setup, striking a balance between usability and performance. SuryaOCR has a more complex installation process because of its deep learning framework and additional requirements, yet it offers REST API support and modular deployment, making it appropriate for scalable enterprise-level applications. Chowdhary et al [1] is simple to install and provides moderate customization options via REST API. Li et al [2] features a complicated installation process due to its TensorFlow dependencies but offers extensive customization and tuning possibilities through Python APIs. Namysl et al [3] is easy to configure and grants moderate customization with API support. Conversely, Mehta et al [6] necessitates integration with natural language processing components, resulting in a more complex setup process, although it provides significant flexibility through it’s a PI.

Table 2. Integration Complexity (Scale of 1 to 5, Lower is Easier)

Model	Ease of Installation	API Support	Customization
SuryaOCR	4	Yes	High
EasyOCR	2	Yes	Moderate
Pytesseract	1	No API	Low
Chowdhary et al [1]	2	Yes	Moderate
Li et al [2]	4	Yes	High
Namysl et al [3]	2	Yes	Moderate
Mehta et al [6]	3	Yes	High

3.4. MULTILINGUAL & SCRIPT SUPPORT

The multilingual and script support accuracy comparison is presented in Fig. 8. SuryaOCR demonstrated the highest multilingual accuracy, particularly for Indic and non-Latin scripts, leveraging transformer-based recognition techniques. Its accuracy for English, Hindi, and Japanese was 98.5%, 94.5%, and 95.2%, respectively. EasyOCR performed well across 80+ languages, though it required fine-tuning for less common scripts. It achieved 96.7%, 89.3%, and 90.7% accuracy for English, Hindi, and Japanese, respectively. Pytesseract supported 100+ languages, but required specific training data for optimal recognition. It had the lowest accuracy among the three, with 94.2%, 85.6%, and 87.3% for English, Hindi, and Japanese. Other models also demonstrated varying levels of multilingual OCR capabilities. Chowdhary et al. [1] model exhibited moderate accuracy levels, achieving 94% in English, 86% in Hindi, and 88% in Japanese. While effective for standard printed text, it faced challenges with handwritten content and complex scripts. Li et al. [2] model leveraged deep learning-based OCR techniques, achieving 97% in English, 92% in Hindi, and 94% in Japanese. Its strong performance was attributed to advanced text normalization techniques and a well-trained transformer-based model, making it highly effective for multilingual applications. Namysl et al. [3] model showed relatively lower accuracy, scoring 93% in English, 85% in Hindi, and 87% in Japanese. It struggled with noisy inputs and script variations but remained a viable option for structured text recognition. Mehta et al. [6] model incorporated NLP-based post-processing, enhancing its recognition capabilities. It achieved

96% in English, 90% in Hindi, and 92% in Japanese, making it a competitive choice for multilingual OCR tasks.

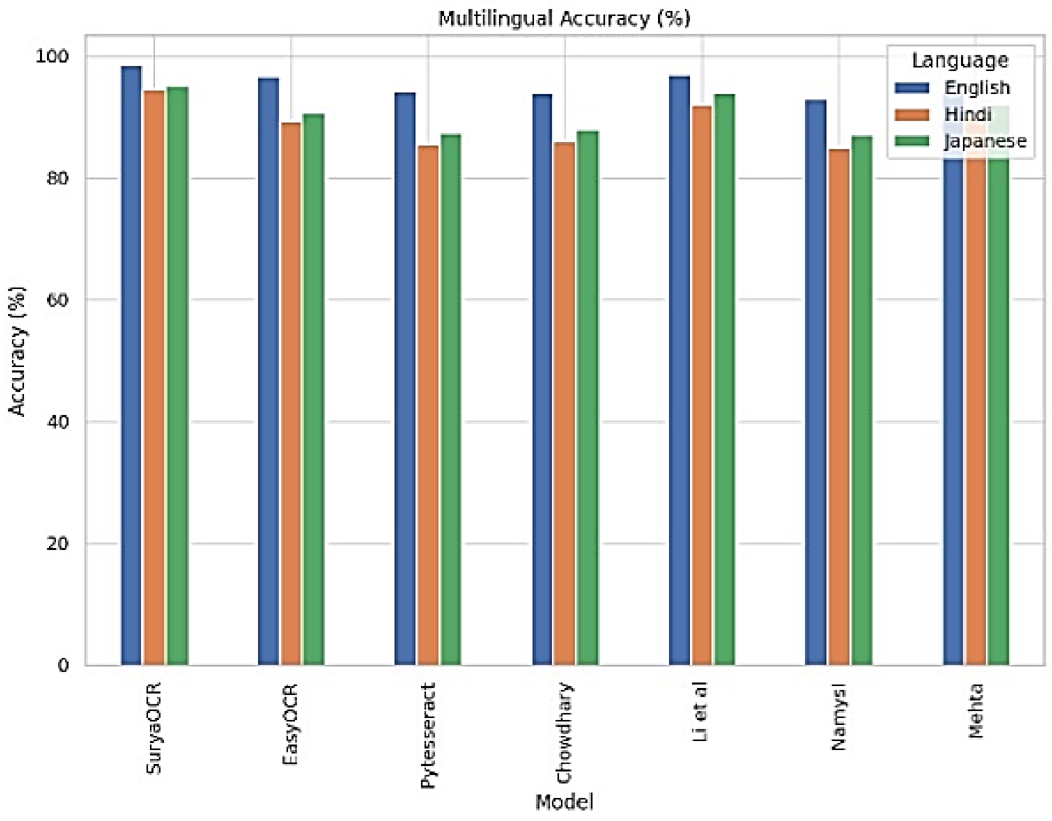


Fig. 8. Multilingual accuracy (%) across key Languages

3.5. HANDLING NOISE AND DISTORTION

Figure 9 shows the comparison of reduction in accuracy for noisy and distorted images. SuryaOCR was best in handling noise environments by utilizing advanced deep learning noise reduction methods which improved recognition accuracy. EasyOCR has difficulties with distortions but delivered good performance on images with minor noise. Pytesseract needed significant tuning steps such as thresholding and binarization to effectively deal with noisy images.

The accuracy reduction of SuryaOCR for blurry text, Rotated text, Low-contrast Image, Text on Complex Backgrounds was 3.5%, 4.2%, 5.1% and 6.8% respectively. EasyOCR with accuracy reduction for blurry text, Rotated text, Low-contrast image, Text on Complex Backgrounds was 7.1%, 8.3%, 9.7%, 11.5% respectively. Pytesseract with accuracy reduction for blurry text, Rotated text, Low-contrast image, Text on Complex Backgrounds was 10.2%, 12.8%, 15.3%, 18.7% respectively. Chowdhary et al [1] with accuracy reduction for blurry text, Rotated text, Low-contrast image, Text on Complex Backgrounds was 6.5%, 7.8%, 9.2%, 11.0% respectively. Li et al [2] with accuracy reduction for blurry text, Rotated text, Low-contrast image, Text on Complex Backgrounds was 4.5%, 5.2%, 6.3%, 7.9%

respectively. Namysl et al [3] with accuracy reduction for blurry text, Rotated text, Low-contrast image, Text on Complex Backgrounds was 7.0%, 8.5%, 10.1%, 12.3% respectively. Mehta et al [6] with accuracy reduction for blurry text, Rotated text, Low-contrast image, Text on Complex Backgrounds was 5.2%, 6.3%, 7.1%, 8.7% respectively.

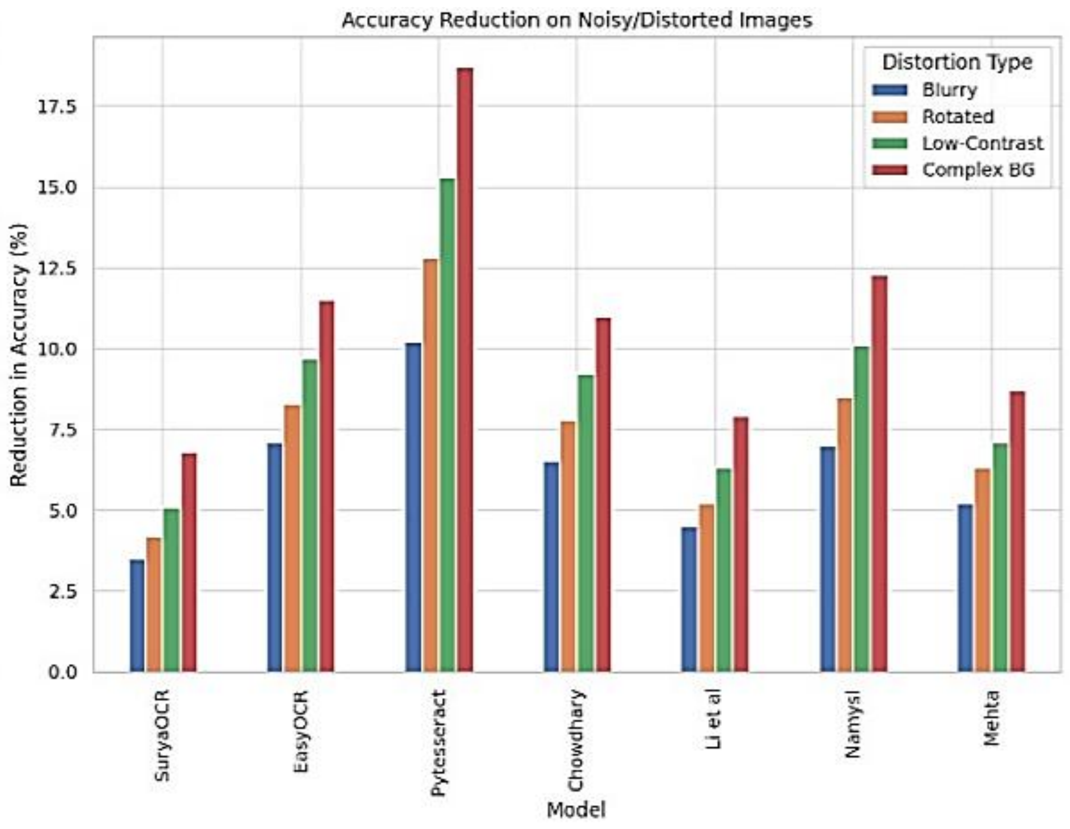


Fig. 9. Accuracy Reduction (%) on Noisy & Distorted Images

4. CONCLUSION AND FUTURE WORK

This research introduces a thorough approach to building an OCR-based assistive system designed to improve accessibility for individuals with visual impairments. By merging software intelligence with hardware feedback mechanisms, we enable users to interpret printed, handwritten, and digital text using tactile and auditory outputs. The system features an array of hardware elements, such as an OLED display, tactile buttons, a camera module, USB keyboard input, and a collection of custom Braille modules operated by electro-mechanical solenoids. These components are effectively coordinated by a Raspberry Pi processing unit and a locally hosted wireless web server, facilitating both manual and remote input. The software aspect of the system is built with dual operation modes–offline (local processing) and online (Wi-Fi-enabled AI-based processing). This allows the solution to adapt to situations where internet connectivity is limited or unavailable. We rigorously evaluated the OCR capability using three advanced models–SuryaOCR, EasyOCR, and Pytesseract–across diverse datasets that included multilingual printed and handwritten text.

Performance metrics such as Character Error Rate (CER), Word Error Rate (WER), processing time, and support for multiple languages were utilized for comparison. SuryaOCR proved to be the most accurate and adept at managing noisy, distorted, and multilingual inputs, despite its more complex setup. EasyOCR offered a good balance between speed and ease of use, while Pytesseract delivered reliable performance on structured text but faced challenges with handwriting and noise. These results provide valuable insights into the appropriateness of different OCR systems for a range of real-world accessibility requirements. From a usability standpoint, the system delivers immediate responses through visual indicators (OLED), auditory alerts (buzzer or speaker), and tactile feedback (Braille). The user interface was crafted to be intuitive and responsive, even without visual guidance. Nonetheless, we recognize that actual usability testing with end-users–especially those with visual impairments–is still required and represents an essential next phase in confirming the practicality and inclusivity of our solution.

The existing prototype utilizes electro-mechanical solenoids for Braille cell activation, which, while functional, are known for being power-intensive and bulky. This impacts the overall portability and energy efficiency of the device. Consequently, a key focus for future versions will be to investigate alternative actuation technologies, such as shape-memory alloys (SMA), piezoelectric actuators, or magnetic/electrostatic microactuators, which could lower power demands and allow for a more compact, wearable design. Additionally, we plan to integrate sophisticated AI-driven post-processing methods (e.g., natural language correction and semantic analysis) to enhance OCR precision, particularly for intricate handwritten or multilingual texts.

Looking forward, we aspire to transform this platform into a more modular, open-source toolkit that can be implemented in educational and public settings, particularly in underserved communities. Our ultimate goal is to help bridge the digital divide by enabling individuals with visual impairments to access, engage with, and contribute to digital content, especially in STEM fields where accessibility resources are still limited. The insights and framework provided in this research establish a foundation for future advancements in intelligent assistive technologies.

REFERENCES

[1] CHAUDHARY K., BALI R., 2021, *EASTER: Efficient and Scalable Text Recognizer*, Proceedings of the 34th Canadian Conference on Artificial Intelligence, https://arxiv.org/abs/2008.07839.

[2] LI M., LV T., CHEN J., CUI L., LU Y., FLORENCIO D., ZHANG C., LI Z., WEI F., 2023, *TrOCR: Transformer-Based Optical Character Recognition with Pre-Trained Models*, Proceedings of the AAAI Conference on Artificial Intelligence, 37/11, 13094–13102, https://doi.org/10.1609/aaai.v37i11.26538.

[3] NAMYSL M., KONYA I., 2019, *Efficient, Lexicon-Free OCR Using Deep Learning*, Proceedings of the 15th International Conference on Document Analysis and Recognition (ICDAR), 339–344, https://doi.org/10.1109/ICDAR.2019.00055.

[4] ISLAM N., ISLAM Z., NOOR N., 2017, *A Survey on Optical Character Recognition System*, Journal of Information and Communication Technology, 10/2, 97–123, https://doi.org/10.48550/arXiv.1710.05703.

[5] SAHU D.K., SUKHWANI M., 2015, Sequence to Sequence Learning for Optical Character, arXiv:1511.04176, https://doi.org/10.48550/arXiv.1511.04176.

[6] RAKSHIT A., MEHTA S., DASGUPTA A., 2023, *A Novel Pipeline for Improving Optical Character Recognition Through Post-Processing Using Natural Language Processing,* Proceedings of the IEEE Guwahati

Subsection Conference (IEEE GCON), 1–6, https://doi.org/10.48550/arXiv.2307.04245.

[7] LEE H.-S., YOON Y., JANG P.-H., CHOI C., 2019, PopEval: A Character-Level Approach to End-To-End Evaluation Compatible with Word-Level Benchmark Dataset, Proceedings of the 15th International Conference on Document Analysis and Recognition (ICDAR), 1211–1216, https://doi.org/10.48550/arXiv.1908.11060.

[8] HWANG M.-Y., SHI Y., RAMCHANDANI A., PANG G., KRISHNAN P., KABELA L., SEIDE F., DATTA S., LIU J., 2023, DISGO: Automatic End-To-End Evaluation for Scene Text OCR, https://doi.org/10.48550/arXiv.2308.13173.

[9] CARRASCO R.C., 2014, An Open-Source OCR Evaluation Tool, Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage (DATeCH), 53–58, https://doi.org/10.1145/2595188.2595221.

[10] NEUDECKER C., BAIERER K., GERBER M., CLAUSNER C., ANTONACOPOULOS A., PLETSCHACHER S., 2021, A Survey of OCR Evaluation Tools and Metrics, Proceedings of the 6th International Workshop on Historical Document Imaging and Processing (HIP), 35–40, https://doi.org/10.1145/3476887.3476888.

[11] LAVRIC A., BEGUNI C., ZADOBRISCHI E., CAILEAN A.M., AVATAMANIEI S.A., 2024, *A Comprehensive Survey on Emerging Assistive Technologies for Visually Impaired Persons: Lighting the Path with Visible Light Communications and Artificial Intelligence Innovations*, Sensors, 24/15, 4834, https://doi.org/10.3390/s24154834.

[12] KATHIRIA P., MANKAD S.H., PATEL J., KAPADIA M., LAKDAWALA N., 2024, *Assistive Systems for Visually Impaired People: a Survey on Current Requirements and Advancements*, Neurocomputing, 606, 128284, https://doi.org/10.1016/j.neucom.2024.128284.

[13] REDDY K.K., BADAM R., ALAM S., SHUAIB M., 2024, IoT-Driven Accessibility: a Refreshable OCR-Braille Solution for Visually Impaired and Deaf-Blind Users Through WSN, J. Econ. Technol., 2, 128–137, 2024. https://doi.org/10.1016/j.ject.2024.04.007.

[14] Al-Salman A., AlSalman A., 2024, Fly-LeNet: a Deep Learning-Based Framework for Converting Multilingual Braille Images, Heliyon, 10/4, https://doi.org/10.1016/j.heliyon.2024.e26155.