EyeNova:Revolutionizing Accessibility With AI-Powered Solutions For The Visually Impaired, Deaf, and Mobility Challenged

EyeNova

01

SMART GLASS

Smart Glasses for the Visually Impaired leverage deep learning and advanced OCR technology to convert text from the physical environment into spoken words. Powered by a Raspberry Pi, these lightweight glasses enable visually impaired individuals to read signs, labels, and documents with ease. With enhanced accuracy and adaptability, the glasses promote independence and confidence in navigating their surroundings.

Problem

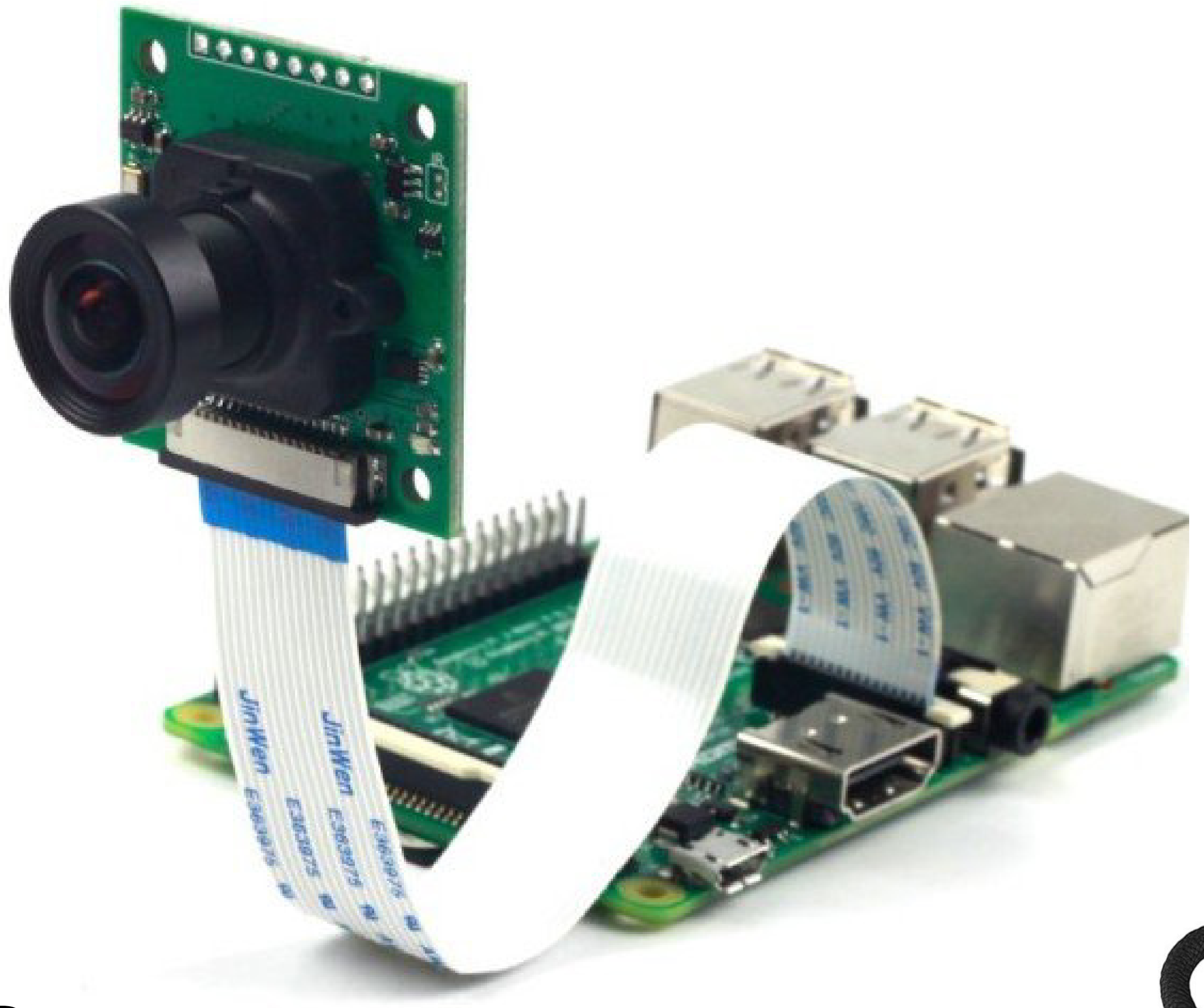
Visually impaired individuals often encounter numerous challenges in reading printed text, which can isolate them and limit their participation in everyday activities. While there are existing solutions, they tend to be either too expensive, large, or complicated for regular use, making them impractical for real-time, on-the-go accessibility.

Proposed Solution

The Smart Glass project is designed to empower users by enabling the real-time conversion of printed text into spoken words. Using OCR and TTS technology integrated into a lightweight and portable design, the Smart Glass offers a user-friendly solution that enhances accessibility and independence for visually impaired individuals.

Requirements

Core Component



Raspberry Pi 4

The central processing unit that runs the OCR and TTS software and handles the overall functionality of the device.

Camera v2

Captures high-resolution images of printed text, which is then processed for text recognition by the OCR.

Speaker

Outputs the audio generated by the TTS engine, providing audible feedback for the user.

Software components

Tesseract OCR: An open-source optical character recognition engine that extracts text from images captured by the camera. It uses deep learning models for text recognition, making it capable of identifying printed text in various fonts and sizes.

pyttsx3 (Text-to-Speech): A text-to-speech engine that converts the extracted text into speech. It works offline, ensuring reliable functionality without internet dependency, and allows customization of speech properties such as rate, volume, and voice.

Software components

GPIO Integration: Uses the Raspberry Pi's General Purpose Input/Output (GPIO) pins to detect button presses, which trigger the capture of images and the processing of text to speech.

Image Preprocessing (Pillow Library): Enhances the captured images before passing them to the OCR engine. This includes resizing the images to a standard resolution and adjusting the quality to improve OCR accuracy.

Threading: Ensures a responsive user experience by handling tasks such as image capture, text processing, and speech output concurrently. This allows the system to run efficiently without lag.

libcamera-still: A command-line tool used to capture images from the Raspberry Pi Camera, integrated into the system to trigger image capture with the press of a button.

Pytesseract

Text Recognition:
Pytesseract enables text extraction from images, handling printed text in scanned images or photos. It supports multiple languages, with additional language packs available for non-English texts.

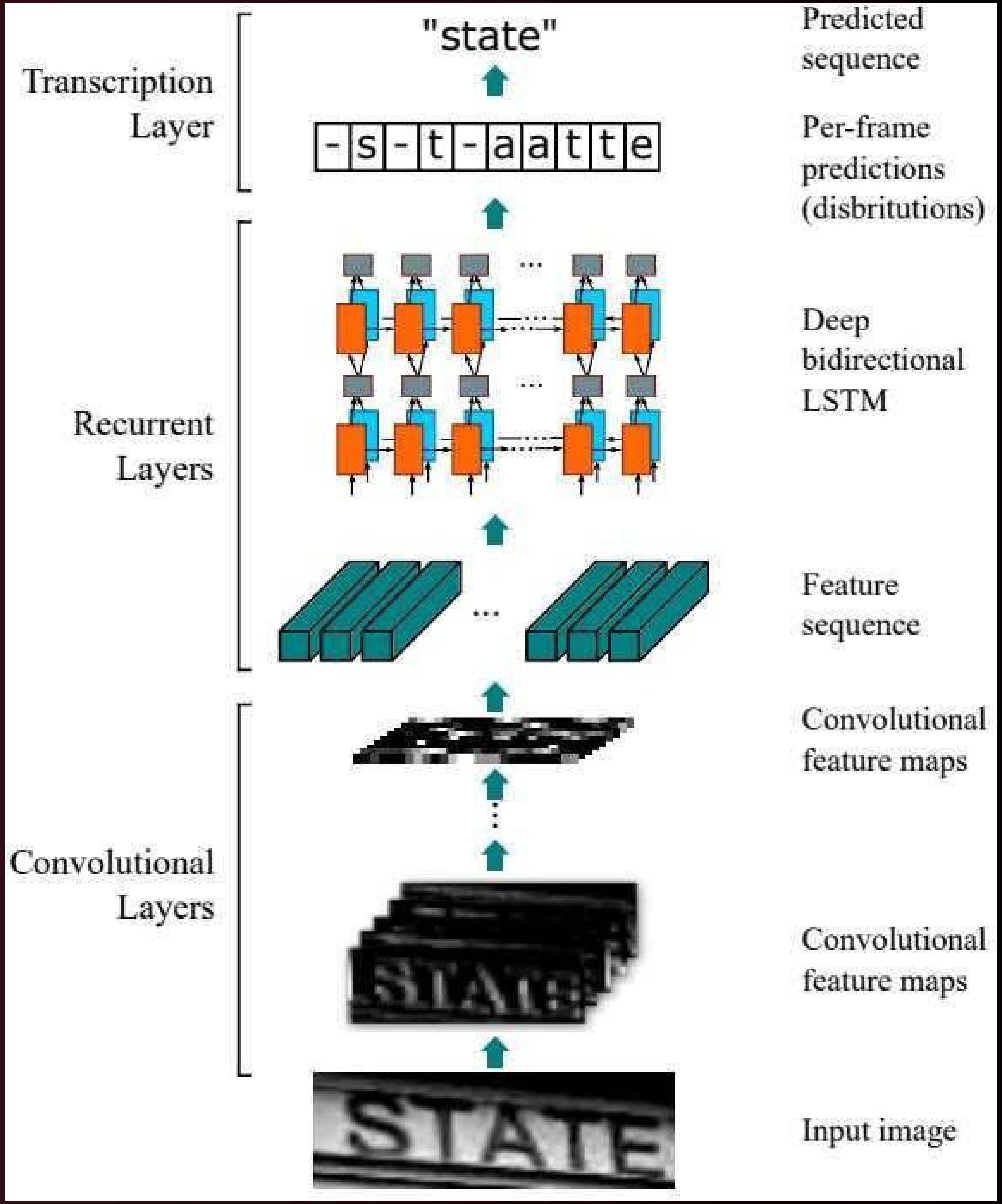
Image Preprocessing:
Preprocessing steps like grayscale conversion, binarization, noise removal, and resizing can enhance OCR accuracy. Pytesseract integrates well with OpenCV for image processing.

Multiple Output Formats:
Pytesseract provides various output formats, including plain text, boxed layout, HOCR (HTML), TSV (Tab Separated Values), and PDF with OCR text overlay.

Multi-language Support:
It can recognize text in various languages, with language selection via the lang parameter. Additional language data files from Tesseract are required for non-English recognition.

Tesseract OCR

The OCR pipeline combines convolutional and LSTM layers for text recognition. Starting with an input image, convolutional layers extract features like edges and textures, which are then organized into a feature sequence. Bidirectional LSTMs process this sequence in both directions to capture character context. Finally, a transcription layer predicts characters per frame, generating the output text, such as "state." This workflow highlights the integration of deep learning to handle text variations and distortions effectively.



```python
import uuid
import subprocess
from PIL import Image
import pyttsx3
from time import sleep
import pytesseract
import RPi.GPIO as GPIO
from threading import Thread
```

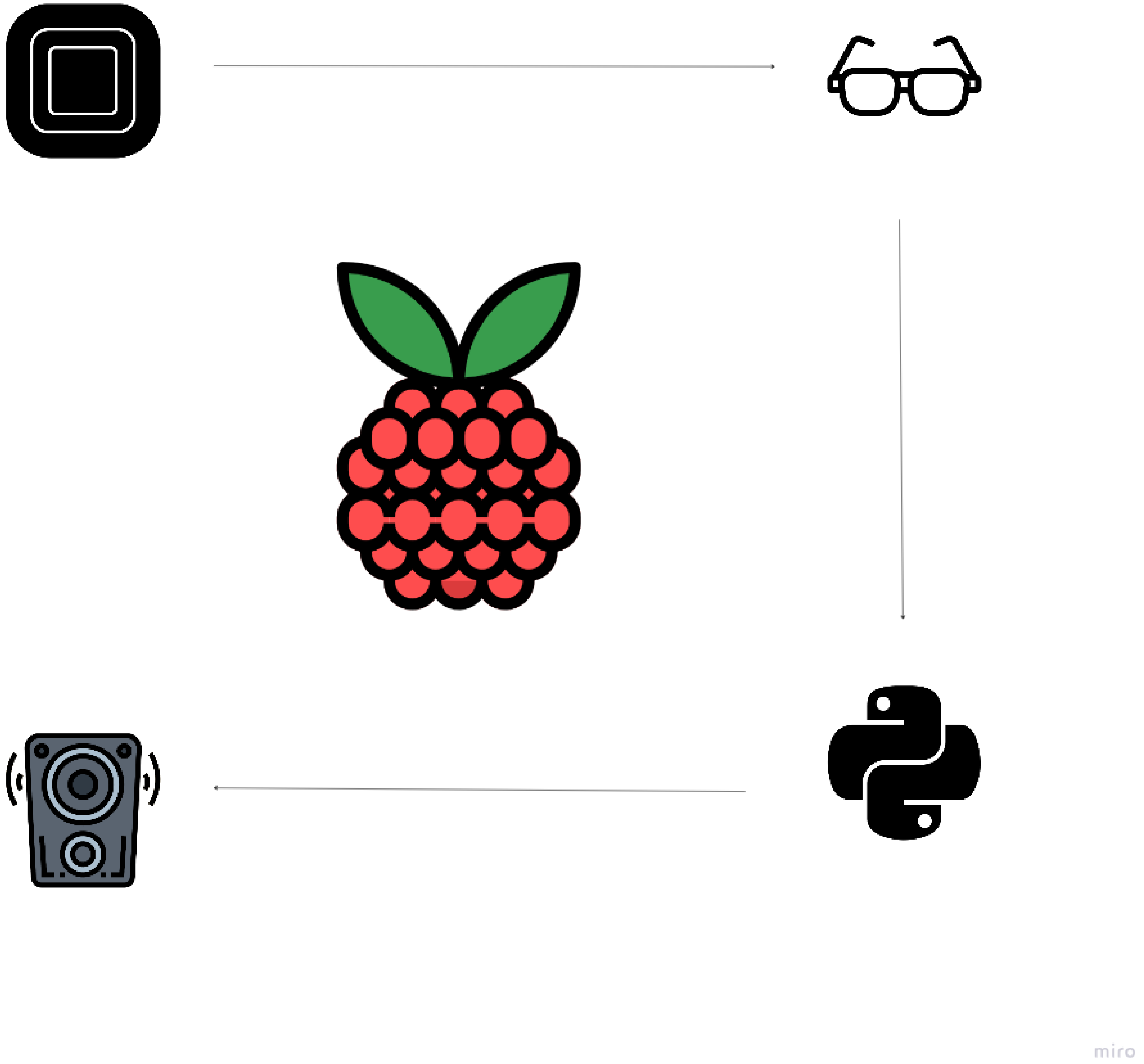
```python
def capture_image():
    img_name = os.path.join(IMAGES_PATH, f'image_{uuid.uuid1()}.jpg')
    try:
        speak_message("Capturing image now.")
        subprocess.run(['libcamera-still', '-o', img_name, '--nopreview'], check=True)
        speak_message("Image captured successfully.")
        return img_name
    except subprocess.CalledProcessError:
        speak_message("Error: Unable to capture the image.")
        return None

```

```
def perform_ocr_and_speak(image_path)
    print(f'Starting OCR on image  {image_path} )
    try:
        # Resize image for better OCR performance
        resize_image(image_path)

        # Perform OCR using pytesseract
        text = pytesseract.image_to_string(image_path)
        print(f'OCR Result \n{text} )

        # Speak the OCR result
        if text strip()
            speak_message( Reading the captured text
                )
            speak_message(text)
        else:
            speak_message( No readable text found in
                the image. )
    except Exception as e
        print(f'Error during OCR or speech conversion
            {e}')
        speak_message( Error processing the image ')
```


miro