DOCUMENTARY

BLINDBOT: VISIONGUIDE ASSIST USING RASPBERRYPI

MODULES

We have completed two kinds of modules: the Face Recognition module and the Online Newspaper Reading module. Both modules were done using Python programming language. After completing these two modules, we are going to integrate them into one module.

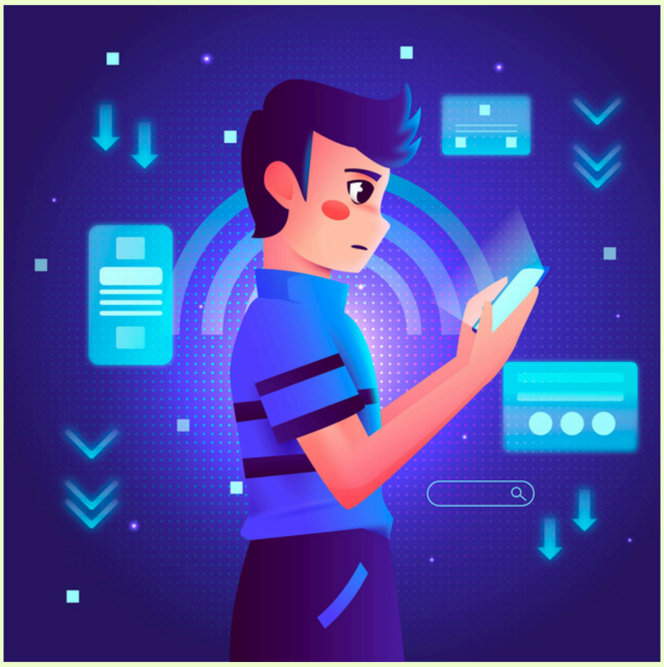
Face Recognition

The Face Recognition module is commonly based on Face Detection framework technology. In this Face Detection framework, we used ANN and dlib's HOG techniques.

An Artificial Neural Network (ANN) is a computational model inspired by the way biological neural networks in the human brain process information.

Dlib's HOG-based face detector uses Histogram of Oriented Gradients (HOG) features to detect faces in images.

Online Newspaper



This code uses a combination of techniques from web data retrieval and JSON Parsing. Let's break down the techniques used:

Web Data Retrieval - This technique involves sending an HTTP GET request to a specified API endpoint to retrieve data.

JSON Parsing - Once the data is retrieved via the HTTP request, it's often in JSON (JavaScript Object Notation) format, which is a lightweight data-interchange format that's easy for humans to read and write, and easy for machines to parse and generate.

CODE EXPLANATION

FACE RECOGNITION

IMPORT LIBRARIES

```
import cv2
import pyttsx3
import dlib
import numpy as np
import face_recognition
import os
import time
```

libraries for facial recognition and text-to-speech functionalities.

- **cv2:** OpenCV library for computer vision tasks.
- **pyttsx3:** Library for text-to-speech conversion.
- **dlib:** Library for facial landmark detection and pose estimation.
- **numpy:** Library for numerical operations.
- **face_recognition:** Library for face recognition tasks.
- **os:** Library for interacting with the operating system.
- **time:** Library for measuring time.

TEXT-TO-SPEECH INITIALIZATION

```
engine = pyttsx3.init()
voices =  engine.getProperty('voices')
engine.setProperty('voice', voices[1].id)
voice (optional)

def speak_name(name):
    engine.say(name)
    engine.runAndWait()
```

Initializes the text-to-speech engine

- **engine.init():** Initializes the text-to-speech engine.
- **voices = engine.getProperty('voices'):** Gets a list of available voices.
- **engine.setProperty('voice', voices[1].id):** Optionally sets the default voice (second voice in the list in this case).
- **speak_name(name) function:** This function takes a name as input and uses the engine to speak it out loud.

FACE DETECTOR INITIALIZATION

```
detector = dlib.get_frontal_face_detector()
```

Initialize Dlib's face detector (HOG-based)

- This detector is used to find faces in an image or video frame.

FACE RECOGNITION MODEL

```
class ANNClassifier:
    def __init__(self):
        self.embeddings_dict = {}
        self.threshold = 0.6

    def load_embeddings(self, name, image_path):
        if not os.path.exists(image_path):
            raise FileNotFoundError(f"Image file
'{image_path}' not found.")

        image =
face_recognition.load_image_file(image_path)
        face_encodings =
face_recognition.face_encodings(image)

  if len(face_encodings) == 0:
     raise ValueError(f"No face detected in image
'{image_path}'.")
embedding = face_encodings[0]
self.embeddings_dict[name] = embedding
```

Defines a ANN (Artificial Neural Network) classifier class for face recognition.

- **ANNClassifier:** This class represents the ANN classifier for face recognition.
- **__init__(self):** The constructor initializes an empty dictionary embeddings_dict to store known face encodings and a threshold value for recognition accuracy.
- **load_embeddings(self, name, image_path):** This function loads a face embedding from an image file for a given name. It checks if the image file exists and extracts the face encoding using face_recognition.load_image_file and face_recognition.face_encodings.

```python
def recognize_face(self, face_encoding):
    min_dis9tance = float('inf')
    recognized_name = "Unknown"

    for name, known_embedding in self.embeddings_dict.items():
        distance = np.linalg.norm(face_encoding - known_embedding)
        if distance < min_distance:
           min_distance = distance
           recognized_name = name

    if min_distance > self.threshold:
        recognized_name = "Unknown"

    return recognized_name
```

- **recognize_face(self, face_encoding):** This function takes a face encoding as input and compares it to the known encodings stored in the embeddings_dict. It calculates the distance between the input encoding and each known encoding. The name associated with the closest (minimum distance) encoding is returned as the recognized name. If the minimum distance is greater than the threshold, "Unknown" is returned.

LOAD KNOWN FACES

```python
ann_classifier = ANNClassifier()
try:
    ann_classifier.load_embeddings("ganesh", "path_to_ganesh_image")
    ann_classifier.load_embeddings("suyamburajn", "path_to_suyamburajan_image")
    ann_classifier.load_embeddings("HarryPotter", "path_to_harry_potter_image")
except Exception as e:
    print(f"Error loading embeddings: {e}")
    exit()
```

Load embeddings for known faces and handle exceptions.

- initializes the ANN classifier and loads the face embeddings for three known individuals: "ganesh", "suyamburajan", and "Harry Potter". If there is an error while loading the embeddings, the program exits.

WEBCAM INITIALIZATION AND TIMER

```python
print("[INFO] sampling frames from webcam...")
engine.say("[INFO] sampling frames from webcam...")
engine.runAndWait()
video_capture = cv2.VideoCapture(0)
duration = 1 * 60
start_time = time.time()
```

- Capture video from the webcam and set a timer for the duration of the program.
- Defines the duration for the facial recognition process (1 minute in this case).

MAIN LOOP

1. TIMER CHECK:

```python
while True:
     elapsed_time = time.time() - start_time
     if elapsed_time > duration:
        print("Timer expired, exiting...")
        engine.say("Timer expired, exiting...")
        engine.runAndWait()
        break
```

- Exit loop if duration has passed.

2. FRAME CAPTURE AND PROCESSING:

```python
ret, frame = video_capture.read()
if not ret:
    print("Failed to grab frame from webcam")
    break
small_frame = cv2.resize(frame, (0, 0), fx=0.5, fy=0.5)
rgb_small_frame = small_frame[:, :, ::-1]
```

- Capture a frame, resize, and convert to RGB.
- Captures a single frame from the webcam.
- Resizes the frame to half its original size for faster processing.
- Converts the frame from BGR color to RGB color, as required by the face_recognition library.

3. FACE DETECTION:

```
faces = detector(rgb_small_frame, 1)
```

- Detects faces in the frame using the Dlib HOG detector.

4. FACE RECOGNITION:

```
for face in faces:
    (x, y, w, h) = (face.left(), face.top(), face.width(), face.height())
    face_encodings = face_recognition.face_encodings(rgb_small_frame, [(y, x + w, y + h, x)])
    if len(face_encodings) > 0:
        face_encoding = face_encodings[0]
        recognized_name = ann_classifier.recognize_face(face_encoding)
    speak_name(recognized_name)
        cv2.rectangle(frame, (x * 2, y * 2), ((x + w) * 2, (y + h) * 2), (0, 255, 0), 2)
        cv2.putText(frame, recognized_name, (x * 2 + 6, (y + h) * 2 - 6), font, 1.0, (255, 255, 255), 1)
```

For each detected face:

- Extracts the face encoding using the face_recognition library.
- Recognizes the face using the ANN classifier.
- Speaks the recognized name using the text-to-speech engine.
- Draws a rectangle around the face and a label with the recognized name.

5. DISPLAY FRAME:

```
cv2.imshow('Camera', frame)
```

- Displays the resulting frame with the detected and recognized faces.

6. EXIT ON 'Q' PRESS:

```
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

- Break loop if 'q' is pressed.

CLEANUP

```
video_capture.release()
cv2.destroyAllWindows()
```

- Release webcam and close windows.

ONLINE NEWSPAPER

IMPORT LIBRARIES

```
import cv2
import json
import pyttsx3
```

- **requests:** Used to make HTTP requests to the news API.
- **json:** Used to parse the JSON response from the API.
- **pyttsx3:** Used for text-to-speech functionality (reading news aloud).

FUNCTION DEFINITION:

```
def get_latest_news(api_endpoint, api_key, category="general", language="en"):
```

- **api_endpoint:** The URL of the news API endpoint.
- **api_key:** Your API key for authentication (replace the placeholder with your actual key).
- **category (optional):** The news category to fetch (defaults to "general").
- **language (optional):** The desired language for articles (defaults to "en").

FUNCTION IMPLEMENTATION:

1. SETTING PARAMETERS

```
    params = {
        "apiKey": api_key,
        "category": category,
         "language": language
    }
```

- Creates a dictionary of parameters to send with the API request.

2. MAKING THE REQUEST

```
    try:
        response = requests.get(api_endpoint, params=params)
        response.raise_for_status()

        data = json.loads(response.text)
        articles = data.get("articles", [])  # Assuming "articles" key holds the data

        return articles
```

- **requests.get:** Sends a GET request to the API.
- **response.raise_for_status():** Raises an error for bad responses.
- **json.loads(response.text):** Parses the JSON response.
- **articles = data.get("articles", []):** Extracts the "articles" list from the response, defaulting to an empty list if not found.

3. HANDLING EXCEPTIONS

```
    except requests.exceptions.RequestException as e:
        print(f"An error occurred while fetching news: {e}")
        return []
```

- Catches any request-related exceptions and prints an error message.
- Returns an empty list if an error occurs.

SETTING UP API KEY AND ENDPOINT

```
YOUR_API_KEY = "641439d0282f485ab91070f692d96eb7"
api_endpoint = "https://newsapi.org/v2/top-headlines?country=in&apiKey=" + YOUR_API_KEY
```

- **YOUR_API_KEY:** Your API key for the news API.
- **api_endpoint:** The endpoint URL for fetching top headlines for India (country=in).

FETCHING LATEST NEWS

```
latest_news = get_latest_news(api_endpoint, YOUR_API_KEY)
```

- Calls the get_latest_news function with the API endpoint and key.

TEXT-TO-SPEECH SETUP

```
if latest_news:
    engine = pyttsx3.init()
    voices = engine.getProperty('voices')
    engine.setProperty('voice', voices[1].id)
```

- Initializes the pyttsx3 engine and sets a voice (e.g., female voice).

READING ARTICLES

```
    num_articles_to_read = 3

     for i, article in enumerate(latest_news):
         if i >= num_articles_to_read:
             break
         title = article.get('title')
         max_description_length = 100
        description = article.get('description')
         shortened_description = description[:max_description_length] + "..." if len(description) > max_description_length else description
         url = article.get('url')

         print(f"Title: {title}")
         print(f"Description: {shortened_description}")
         print(f"URL: {url}")
         print("-" * 50)

         engine.setProperty('rate', 140)

         engine.say(f"Title: {title}")
         engine.say(f"Description: {shortened_description}")
         engine.runAndWait()
```

- num_articles_to_read defines the number of articles to process (adjustable).

A for loop iterates through each article in latest_news:

- enumerate(**latest_news)** helps access both the index (i) and the article itself in each iteration.
- The loop breaks after processing the specified number of articles.
- **article.get('title')** extracts the article title (assuming it exists under the "title" key).
- The description is retrieved (**article.get('description')**) and shortened to a maximum length (**max_description_length)** with an ellipsis (...) if it exceeds the limit.
- The article URL is obtained **(article.get('url'))**.
- Information about the article (title, shortened description, URL) is printed.
- A separator line is printed for clarity.
- The speech rate is adjusted (optional).
- **engine.say** speaks the title and shortened description aloud.
- **engine.runAndWait** waits for the speech to finish.

ERROR HANDLING

```
else:
    print("Error: Could not retrieve latest news.")
```

- Prints an error message if no news articles are retrieved.

INTEGRATED MODULE

Now we have integrated these two modules based on speech recognition. If we say 'run the face module,' it will run the face recognition module, and if we say 'run the newspaper module,' it will run the newspaper module. If we say 'exit from the code,' it will exit from the code.

In this, speech recognition techniques use the Google Web Speech API to convert spoken words into text and adjust the recognizer's sensitivity to ambient noise.

IMPORT LIBRARIES

```
import speech_recognition as sr
import pyttsx3
import face_recognition
import cv2
import numpy as np
import dlib
import sys, time
import requests
import json
import time
import os
```

- **speech_recognition** is used for converting speech to text.

DEFINING FUNCTIONS

```
def perform_face_recognition():
  Here is the code for the face recognition module.

def perform_newspaper_recognition():
  Here is the code for the online newspaper module.
```

- **perform_face_recognition:** A placeholder function that runs the face recognition module
- **perform_newspaper_recognition:** A placeholder function that runs the online newspaper module

INITIALIZATION AND DEFINING RESPONSE MAPPING AND FLAGS

```
r = sr.Recognizer()
engine = pyttsx3.init()

response_map = {
  "run the face module": "fine",
  "run the newspaper module": "fine",
  "exit from the code": "ok",
}

face_recognition_done = False
```

- **r = sr.Recognizer():** Creates a recognizer object to handle speech recognition.
- **engine = pyttsx3.init():** Initializes the text-to-speech engine for voice output.
- **response_map:** This dictionary maps user phrases to predefined responses.
- "**run the face module**": Triggers face recognition (not implemented yet).
- "**run the newspaper module**": Triggers newspaper recognition (not implemented yet).
- "**exit from the code**": Exits the program.
- **face_recognition_done = False:** A flag to keep track of whether face recognition has been performed.

DEFINE SPEAKTEXT FUNCTION

```
def SpeakText(text):

  engine.say(text)
  engine.runAndWait()
```

- **SpeakText:** A function that takes text as input and speaks it using the text-to-speech engine.

MAIN LOOP

```
while True:
   try:
       with sr.Microphone() as source:
      r.adjust_for_ambient_noise(source, duration=0.2)
         audio = r.listen(source)
         text = r.recognize_google(audio).lower()
         print("You said:", text)
```

- Start an infinite loop to continuously listen for user input.
- Use a try-except block for error handling.
- Use the microphone as the audio source.
- Adjust the recognizer sensitivity to ambient noise for 0.2 seconds.
- Listen to the audio from the microphone.
- Recognize and convert the audio to text using Google's speech recognition service and convert it to lowercase.
- Print the recognized text.

HANDLING RECOGNIZED TEXT

```
if text in response_map:
        if text == "run the face module" and not face_recognition_done:
            message = perform_face_recognition()
                    face_recognition_done = True

        if text == "run the newspaper module" and not face_recognition_done:
            message = perform_newspaper_recognition()
            face_recognition_done = True

        if text == "exit from the code" and not face_recognition_done:
            print("Exiting program")
            SpeakText("Exiting program")
            break

        else:
            response = response_map[text]
        else:
            print("------- ------- ------- -------")
            SpeakText("try again")
```

- Check if the recognized text matches any key in response_map.
- If the text is "run the face module" and face recognition hasn't been performed yet, call perform_face_recognition and set face_recognition_done to True.
- If the text is "run the newspaper module" and face recognition hasn't been performed yet, call perform_newspaper_recognition and set face_recognition_done to True.
- If the text is "exit from the code", print and speak the exit message, then break the loop to terminate the program.
- If the text does not match any custom response, prompt the user to try again by speaking "try again".

ERROR HANDLING

```
except sr.RequestError as e:
    print("Could not request results; {0}".format(e))

except sr.UnknownValueError:
    print("Could not understand audio")

  face_recognition_done = False
```

- Handle errors where the speech recognition request fails.
- Handle errors where the recognizer cannot understand the audio.
- Reset the face_recognition_done flag at the end of each loop iteration to allow for new commands.

NOW WE ARE GOING TO USE THIS INTEGRATED CODE IN A RASPBERRY PI.

ABOUT RASBERRY PI 5

Processor: Quad-core Broadcom BCM2712 ARM Cortex-A76 64-bit processor clocked at 2.4 GHz, delivering 2-3 times the efficiency of the Raspberry Pi 4.
Memory: 4 GB of LPDDR4 RAM.
Display and USB Ports:

- Two micro HDMI connectors.
- Two USB 3.0 ports.
- Two USB 2.0 ports.
- USB-C power connector.

Camera and Display Interfaces:

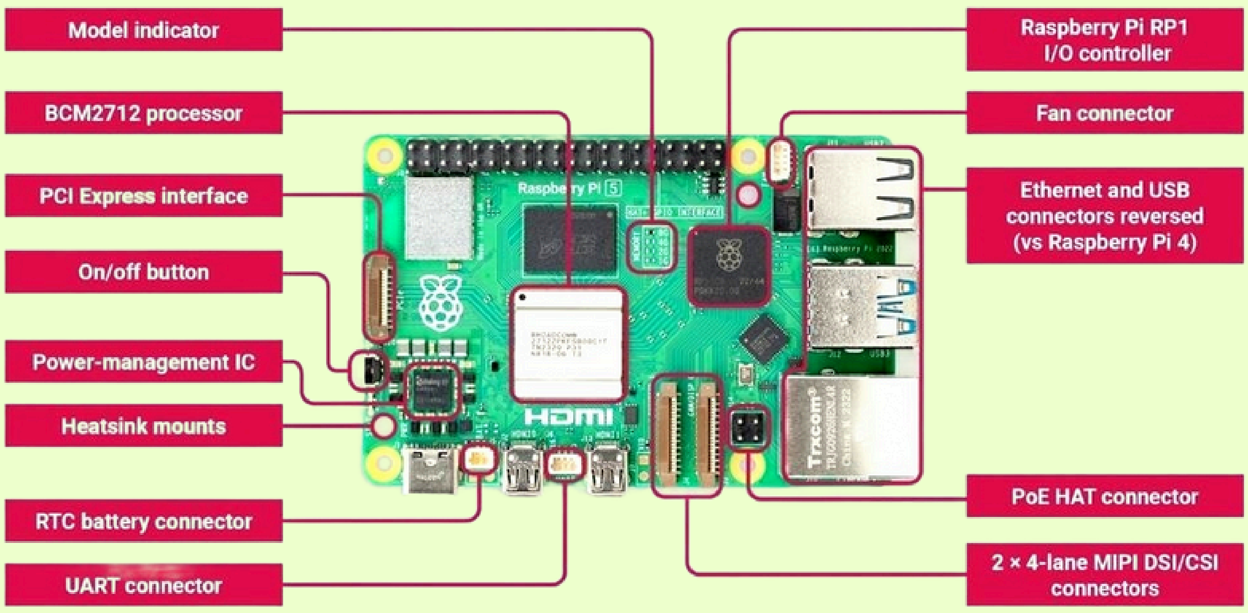
- Two interfaces for a camera.
- MIPI DSI/CSI display interface.

Expansion and Connectivity:

- PCI Express 2.0 connector.
- Dual-band WiFi (2.4 GHz and 5 GHz).
- Bluetooth 5 BLE.
- Fast 1000 Mb/h Ethernet port with PoE capability using the PoE+ HAT overlay.

Additional Features:

- Space for a fan.
- UART connector.
- Space for an RTC battery.
- Dedicated power button.
- Standard 40-pin GPIO connector (2×20 pin, 2.54 mm pitch).
- MicroSD card slot for the operating system.
- Mounting holes for an official radiator with a fan.



- In our Raspberry Pi 5, we downloaded PyCharm and ran the integrated code. Before that, we used more libraries, so we needed to download all the necessary libraries. One more thing to note is that the Raspberry Pi typically runs a Linux-based operating system. The most common OS for Raspberry Pi is Raspberry Pi OS (formerly known as Raspbian), which is a Debian-based operating system optimized for the Raspberry Pi hardware.
- We need to have pip installed on our system. Pip is a package manager for Python that allows you to install and manage additional libraries and dependencies that are not distributed as part of the standard library.

For Linux-based systems, you can usually install pip using your package manager in the command terminal.

- `sudo apt update`
- `sudo apt install python3-pip`
- `pip install --upgrade pip`

Once pip is installed, you can use it to install the required libraries:

- `pip install speechrecognition`
- `pip install pyttsx3`
- `pip install face_recognition`
- `pip install opencv-python`
- `pip install numpy`
- **`pip install dlib`**
- `pip install requests`

dlib has some additional dependencies, particularly when installing on systems like Linux or macOS. Before installing dlib, We need to install some prerequisite packages:

Build tools and libraries:

- `sudo apt-get update`
- `sudo apt-get install build-essential cmake`

Boost and other required libraries:

- `sudo apt-get install libgtk-3-dev libboost-all-dev`

Python development headers:

- `sudo apt-get install python3-dev`

Additional dependencies for dlib:

- `sudo apt-get install libopenblas-dev liblapack-dev`

Now, when we run the code, the output comes without errors. Additionally, we must have a separate microphone, webcam, speaker, and stable internet connection.