Zagazig university
Faculty of Engineering
Computer and Systems department

Assistive technologies for the visually impaired

GROUP 9

Under supervision of
DR: MIRA MAGDY

Project member:

Esraa Samir Ezat Sadik
Asmaa Mohamed Mustafa Abdel-Aziz
Dina Ashraf mohamed
Mohamed Ebrahim Hamdy El-Gebaly
Mohamed Sameh Mohamed Talaat
Mohamed Abdel-Sattar Abdel-Latif
Mahmoud Khaled Ahmed Abdel-Wahab
Mai Alaa Khamis Mohamedeen
Yasmin Mohamed Mohamed Selmi

Table of Contents

List of Reference

1. **https://github.com/ultralytics/ultralytics**
2. **https://github.com/pedroprates/mobile-face-net**
3. **https://arxiv.org/ftp/arxiv/papers/1804/1804.07573.pdf**
4. **https://www.mi-research.net/article/doi/10.1007/s11633-023-1423-y**
5. **https://github.com/ShiqiYu/libfacedetection.train**
6. **https://docs.videosdk.live/flutter/guide/video-and-audio-calling-api-sdk/concept-and-architecture**
7. **https://github.com/videosdk-live/videosdk-rtc-flutter-sdk-example.git**
8. **https://developers.google.com/maps/documentation/**
9. **https://docs.flutter.dev/accessibility-and-localization/internationalization**

Abstract

The visually impaired face numerous challenges in their daily lives, from recognizing faces and banknotes to identifying colors and reading text. To address some of these challenges, this project proposes a novel system that consists of smart glasses and a mobile app. The smart glasses are equipped with a compute module and a camera that enable them to perform various functions, including face recognition, banknotes recognition, color recognition, reading text, and object detection. The glasses use a combination of computer vision algorithms and machine learning models to process the visual information and provide audio feedback to the user.

The mobile app serves as a hub for the smart glasses, allowing the user to connect to the internet, access additional functionality, and customize the settings. The app also features a chatgpt plugin that uses speech-to-text and text-to-speech technologies to facilitate natural language interaction with the user. The chatgpt plugin is designed to provide assistance and information on a wide range of topics and domains.

The proposed system has the potential to enhance the independence and interaction of the visually impaired, enabling them to perform daily tasks more efficiently and with greater confidence.

Chapter 1

Introduction

1.1. Background

Visual impairment affects millions of people across the globe. According to the World Health Organization (WHO) There are over 285 million people worldwide who are visually impaired, and this number is expected to grow in the coming years. Visual impairment can be caused by a variety of factors, including birth defects, diseases, and injuries. People with visual impairments can face a number of challenges in their daily lives, including difficulty navigating their surroundings, reading and writing, and interacting with others. making everyday tasks challenging and limiting independence. However, recent rapid advances in computer vision, machine learning have enabled the development of electronic assistive technologies that can potentially restore functional vision for the visually impaired. Although some solutions exist to address the same problems, they either lack functionality or are impractically expensive and not easily accessible.

1.2. Problem Statement

The problem of accessibility for the visually impaired has been a longstanding issue that requires urgent attention. The current solutions available in the market aimed at solving this problem are either too expensive or lack the required functionality. This makes it difficult for visually impaired individuals to lead independent and productive lives.

Many of the existing assistive technologies are prohibitively expensive, making them inaccessible to a significant proportion of visually impaired individuals who may not have the financial means to afford them. Additionally, even for those who can afford them, the functionality of these products may not meet their specific needs. For example, some products may provide OCR and text-to-speech capabilities, but lack other features.

This lack of affordability and functionality in existing assistive technologies limits the ability of visually impaired individuals to perform various day-to-day activities independently. This includes tasks such as reading, identifying people, and navigating their surroundings. As a result, visually impaired individuals are often dependent on others for assistance, which can lead to a loss of independence and reduced quality of life.

Therefore, there is a need for an affordable assistive technology solution that provides comprehensive functionality to visually impaired individuals. This project aims to address this need by developing a smart glasses and mobile app system that is affordable, accessible, and offers a wide range of functionalities to improve the lives of visually impaired individuals.

1.3. Proposed Solution

In this project, we propose assistive technologies for addressing daily problems facing visually impaired individuals. The solution should be affordable while providing the required functionality to address the problems they face in their day-to-day lives.

Assistive technologies for the Visually Impaired are an innovative solution that combines the power of wearable technology and a mobile app designed to help individuals with visual impairments to navigate their surroundings with greater ease and independence.

The proposed Smart Glasses will be equipped with a high-resolution camera that captures real-time images of the user's surroundings, which will be processed using computer vision algorithms to identify objects, people. The glasses will use audio to provide users with information about their surroundings.

In addition to object recognition, the Smart Glasses will also incorporate facial recognition technology to help visually impaired users recognize and distinguish between individuals. This feature will be particularly useful for social interactions, allowing users to identify friends and family members with ease.

The Smart Glasses will also have the ability to recognize and distinguish between different banknotes, enabling visually impaired users to handle money with greater confidence and independence.

The glasses will use color recognition technology to identify different colors, which will be particularly useful for distinguishing between clothing and other objects.

Furthermore, the Smart Glasses will include an OCR (Optical Character Recognition) and text-to-speech features that can read printed text aloud to the user, enabling them to access written information with greater ease. The glasses will have the ability to recognize and read a range of text, including books, signs, and menus.

- The accompanying mobile app serves as a hub for the smart glasses, connecting them to the internet and unlocking additional functionalities. The app also features a cutting-edge chatbot technology powered by ChatGPT, which takes input through speech-to-text from the user and responds with text-to-speech output. The app also features a Text Recognition that enable visually impaired users to access and comprehend textual content that may otherwise be inaccessible to them.also

the app provided with probability of making a video call and get his location using voice. Also, The application provides support for two languages, Arabic and English, This allows users to seamlessly switch between the two languages based on the language of their mobile settings, ensuring a localized and user-friendly experience.
This component enhances the user experience by providing an intuitive interface for the visually impaired to get high quality information and interact with their environment and communicate with others.

Overall, this project aims to develop an advanced and user-friendly Smart Glasses system that can significantly enhance the quality of life for visually impaired individuals. The proposed design represents a significant step forward in the field of assistive technology and has the potential to revolutionize the way visually impaired individuals navigate and interact with the world around them.

Chapter 2
Literature Review

In this chapter, a review of past work and research related to the project is examined. The project looks at many areas, first the needs and expectations of the visually impaired people are reviewed, followed by a discussion about some examples of the available and suggested aids and technologies to assist the visually impaired in their lives.

2.1. The needs and expectations of the visually impaired people

In order to design an assistive device for the visually impaired, information about their needs and their expectations should be gathered for a better design, it is easy to understand that the main problem is the sight loss, but to explore the situation without sight and to identify their needs is not an easy task for the sighted people. Many needs have been identified by many papers including papers written by visually impaired people to help the researchers to understand the issues better. as difficult as it seems to classify the information needs for visually impaired; most papers has identified some types of information needs for a defined task which are: the function of the task, the form in which we find things related to the task, the clusters they belong to, the agents who initiate it, the users and finally the information about the mechanism used to find things. The depth of the details required by the visually impaired is also important, a paper written by one of the visually impaired people called pictures into words discussed the blind imagination, and that the amount of information needed actually depends on the personality of the person and his history, people who were having normal 6 vision for a long time might be eager to know more details than those who were born without vision. The

need for education is a major need for every human being; it is a basic human right that is contributing toward the development of each and every one of us. Inclusive education is a case study that has been researched a lot lately. It means that “all students attend and are welcomed by their neighborhood schools in age-appropriate, regular classes and are supported to learn, contribute and participate in all aspects of the life of the school.” . UNESCO has counted some of the problems that are blocking inclusive education from being applied nowadays, the cultural and social background that affect the perception about people with disabilities and therefore negatively affecting the visually impaired and leading to isolation, also classifying students depending on their type of disability in certain groups with certain amount of information and tools which limit their education level and therefore their ability to develop their best in term of career and social life which result in low level of income. Those problems can be eliminated by the use of suitable technology in modern education and the inclusive education techniques which until now are not sufficiently applied.

2.2. Existing Products

This section is an overview of the existing products that aim to solve the problems facing the visually impaired individuals. The products we highlight include the following:

2.2.1 OrCam MyEye

2.2.2 Aira Envision Glasses

2.2.3 eSight 3 (Headset)

2.2.4 ThirdEye

2.2.5 Oton glasses

2.2.1 OrCam MyEye

- OrCam developed the MyEye device, which is a compact smart camera mounted on eyeglasses. Created by OrCam Company founded by Amnon Shashua and Ziv Aviram. There are 2 types of products, one of them is **OrCam MyEye Smart** has just one feature and the other is **OrCam MyEye Pro** has all features.
- Features
    - Read Text: The device's precision laser targeting captures & reads any text you choose.
    - Smart Reading: Request text of interest, and the relevant information is read to you.
    - Recognize Faces: Know who’s in front of you.
    - Identify Products: Instantly identify consumer products and read barcodes.
    - Money Notes: Verify the denominations of paper currency.
    - Color Recognition: Identify colors from surfaces, such as clothing.
- **Cost**
    - **OrCam MyEye Smart $3500**
    - **OrCam MyEye Pro $4250**

2.2.2 Envision Glasses

- Envision Glasses are AI-powered smart glasses with an integrated camera and built-in speakers that speak out the visual world.
- Features:
    - Instant Text
    - Scan Text
    - Batch Scan
    - Call an Ally & Call Aira
    - Describe Scene
    - Detect Light
    - Recognize Cash
    - Detect Color
    - Find People
    - Find Objects
    - Teach a Face and Explore

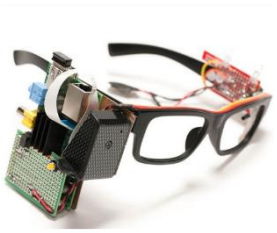
- **Cost: $2499**

Env

2.2.3 eSight 3 (Headset)

- Overview: eSight's smart glasses use high-resolution cameras and advanced image processing to provide enhanced vision for individuals with low vision. It captures and displays real-time video, helping users see and navigate their surroundings more effectively. Created by eSight Corporation Company
- Features:
    - Enhanced Vision: It captures high-resolution video in real-time, enhances the image quality, and displays it on the headset's screens.
    - Increased Independence: It allows users to engage in various activities such as reading, performing daily tasks, and navigating their surroundings.
    - Versatility and Flexibility: The eSight 3 headset is designed to be lightweight and portable, allowing users to use it in various settings and activities.
    - User-Friendly Interface: The eSight 3 headset has an intuitive interface that is easy to use.
- **Cost: $6950**

2.2.4 Oton Glasses



- Overview: Created by a Japanese company (Keisuke Shimakage). Smart glasses are designed to help dyslexics to read. The camera will capture pictures of words that the user wants to read and read the words for the user via the earpiece.
- Features :

  - Only used for reading.
  - The glasses are going to convert symbols into sounds.
  - The glasses currently feature a conversion of English and Japanese text into voice audio and translate multilingual text into Japanese and English voice audio.

- **Cost: 50$**

2.3. Comparison

This section includes a comparison between existing products and our proposed system in terms of price and features of each of them.

Device	**Features**	**Price**	**Pros**	**Cons**
OrCam MyEye Smart	only reading text (one task)	3500$	effective at reading	limited functionality unaffordable
OrCam MyEye Pro	multiple features including reading (multiple tasks)	4200$	multiple functionalities	unaffordable
Envision Glasses	multiple features including reading (multiple tasks)	2500$	multiple functionalities	unaffordable
eSight 3	multiple features including reading (multiple tasks)	6900$	multiple functionalities	unaffordable
Oton Glasses	only reading text (one task)	50$	affordable	limited functionality and language support
Proposed System	multiple features including reading with mobile app that has additional features	<150$	affordable multiple functionalities	current design isn’t production ready

Table 1. Comparison between proposed design and available assistive devices

Chapter 3
Project Requirements

3.1. Software Requirements

3.1.1 Operating System: Raspberry Pi OS Lite

Raspberry Pi OS Lite is a lightweight operating system designed for use on the Raspberry Pi single-board computer. It is a minimalist version of the full Raspberry Pi OS, optimized for users who want an operating system that doesn't come pre-installed with a lot of unnecessary software.

It is ideal for projects that require a low-overhead operating system that can be customized to suit the user's needs. With Raspberry Pi OS Lite, users have full control over their system and can add only the software packages they need, making it a popular choice for hobbyists, makers, and developers.

The OS is based on the Debian operating system and includes a range of powerful tools and utilities that make it easy to configure and manage the Raspberry Pi device.

Overall, Raspberry Pi OS Lite is a versatile, flexible, and lightweight operating system that provides a solid foundation for a wide range of Raspberry Pi projects.

3.1.2 Programming language: Python (version 3.9.2)

Overview

Python is a popular programming language. It was created by Guido van Rossum and released in 1991. It is a high-level, interpreted programming language known for its simplicity and readability. Python emphasizes code readability and uses indentation instead of braces, which makes it easy to understand and write.

Python is a versatile language that supports multiple programming paradigms, including procedural, object-oriented, and functional programming. It has a large and active community of developers, which has contributed to the growth and popularity of the language.

Features of python

- **Easy to Learn**:

  Python's syntax is designed to be straightforward and easy to understand, making it an excellent language for beginners. Its readability helps reduce the learning curve and allows developers to write code quickly.

- **Interpreted Language**:

  Python is an interpreted language, meaning that it does not need to be compiled before execution. This allows for rapid development and easy debugging as you can execute Python code directly.

- **Dynamic Typing**:

  Python uses dynamic typing, which means you don't need to declare variable types explicitly. Variable types are determined at runtime, making Python more flexible and allowing for faster development.

- **Extensive Standard Library**:

Python comes with a comprehensive standard library that provides a wide range of modules and functions for various tasks. It includes modules for file I/O, networking, web development, regular expressions, and much more. The standard library reduces the need for external dependencies in many cases.

- **Large Ecosystem of Third-Party Packages**:

  Python has a vast ecosystem of third-party packages and libraries that extend its capabilities. These packages cover a wide range of domains, such as data analysis, machine learning, web development, scientific computing, and more. Popular packages include NumPy, Pandas, TensorFlow, Django, Flask, and Matplotlib, to name just a few.

- **Cross-Platform Compatibility**:

  Python is available on multiple platforms, including Windows, macOS, and various Linux distributions. This allows developers to write code once and run it on different operating systems without significant modifications.

- **Community and Documentation**:

  Python has a vibrant and supportive community of developers who contribute to its growth. The Python community actively shares resources, tutorials, libraries, and frameworks, making it easier to learn and solve problems. Python also offers extensive official documentation, which provides guidance and explanations for its various features.

3.1.3 Framework: PyTorch Machine learning framework (version 1.13)

Overview

PyTorch is a popular open-source machine learning framework for Python that is widely used for deep learning tasks. It provides a flexible and efficient way to build and train neural networks and other machine learning models.

Features

- **Dynamic Computation Graph**:

PyTorch uses a dynamic computation graph, which means that the graph is built on the fly during runtime. This allows for more flexibility and ease in model development, as you can change the model's structure and behavior dynamically.

- **Pythonic Nature**:

Python is a popular and versatile programming language, and PyTorch leverages its strengths. With PyTorch, developers can write code in a familiar, Pythonic style, taking advantage of the language's expressive features. This facilitates code readability, maintainability, and encourages rapid experimentation.

- **Efficient GPU Acceleration**:

PyTorch provides native GPU support, enabling efficient computation on NVIDIA GPUs. This GPU acceleration is crucial for training large deep learning models, as it significantly speeds up the training process. PyTorch's integration with GPUs allows for seamless utilization of their parallel processing capabilities, leading to faster training and inference times.

- **Research-Oriented Framework**:

PyTorch has gained popularity among researchers due to its dynamic nature, which allows for more flexibility in experimentation. Many state-of-the-art

research papers in various domains, such as computer vision and natural language processing, use PyTorch as their preferred deep learning framework. The flexibility of PyTorch enables researchers to quickly implement and test new ideas.

- **Industry Adoption**:

PyTorch has also seen increasing adoption in industry applications. Its flexibility, ease of use, and strong performance have made it a popular choice for developing production-grade deep learning models. Many companies and organizations, ranging from startups to large enterprises, have adopted PyTorch for tasks such as computer vision, natural language processing, recommendation systems, and more.

- **Tensor Operations**:

PyTorch provides a powerful library called "torch" for tensor computations. Tensors are similar to NumPy arrays but with GPU acceleration support. PyTorch tensors enable efficient numerical operations and automatic differentiation, which is essential for training neural networks.

- **Neural Network Building Blocks**:

PyTorch offers a rich set of building blocks for creating neural networks, including various types of layers, activation functions, loss functions, optimizers, and more. These components can be easily combined to build and customize complex models.

- **GPU Support**:

PyTorch has native GPU support, which allows for efficient computation on NVIDIA GPUs. This enables accelerated training and inference for deep learning models, improving performance significantly.

- **Integration with Python Ecosystem**:

PyTorch seamlessly integrates with the broader Python ecosystem, including popular libraries such as NumPy, pandas, and scikit-learn. This integration allows for smooth data preprocessing, analysis, and visualization alongside PyTorch's deep learning capabilities.

- **Large Community and Active Development**:

PyTorch has a large and vibrant community of developers, researchers, and enthusiasts. This active community contributes to the continuous development of PyTorch, with regular updates, bug fixes, and the introduction of new features.

PyTorch is widely used in various domains, including computer vision, natural language processing (NLP), reinforcement learning, and more. It has been adopted by both researchers and industry practitioners for its flexibility, performance, and ease of use.

It's worth mentioning that PyTorch is often compared to TensorFlow, another popular deep learning framework. While both frameworks have similar functionalities, PyTorch is often favored for its dynamic graph and intuitive programming interface, while TensorFlow's static graph offers benefits in deployment and production scenarios.

Whether you're a beginner or an experienced deep learning practitioner, PyTorch provides a powerful and versatile framework for developing and training machine learning models.

3.1.4 NumPy for numerical computing (version 1.20)

Overview 

NumPy (Numerical Python) is a Python library that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. It is one of the core libraries for numerical computing in Python and is widely used in various scientific and data analysis applications.

Features

- **ndarray**:

The nd array (N-dimensional array) is the core data structure in NumPy. It represents multi-dimensional arrays of homogeneous data types and provides efficient storage and operations on these arrays. Ndarrays offer a fast and memory-efficient way to store and manipulate large amounts of numerical data.

- **Vectorized Operations**:

NumPy enables vectorized operations, which allow you to perform element-wise operations on arrays without the need for explicit loops. This significantly improves the performance and readability of numerical computations. NumPy functions are designed to operate on entire arrays, making it convenient for mathematical and scientific calculations.

- **Efficient Array Operations**:

NumPy provides a vast collection of functions for array operations, such as mathematical, logical, statistical, and linear algebraic operations. These functions are optimized for performance and often leverage highly efficient low-level implementations, resulting in faster computations compared to traditional Python loops.

- **Integration with Python**:

NumPy seamlessly integrates with the Python programming language, making it easy to incorporate numerical computing capabilities into Python applications. NumPy arrays can interact with other Python libraries and data structures, allowing for seamless data exchange and analysis.

3.1.5 OpenCV for common computer vision functions (version 4.6.0)

Overview

OpenCV (Open-Source Computer Vision) is widely used with Python due to its Python bindings, which provide a convenient and efficient way to leverage the functionalities of OpenCV within Python applications. OpenCV's Python interface allows developers to utilize its computer vision algorithms and image processing capabilities with the simplicity and flexibility of the Python programming language.

Features

- **Installation**:

You can install OpenCV for Python using popular package managers like pip or conda. The installation process typically involves installing the OpenCV package and its dependencies, which can be done with a simple command.

- **Image and Video Input/Output**:

OpenCV's Python interface allows you to read and write images and videos using familiar Python syntax. You can use functions like cv2.imread() to read

images, cv2.imwrite() to save images, and cv2.VideoCapture() to read and process videos.

- **Image Processing and Manipulation**:

OpenCV provides numerous image-processing functions that are accessible in Python. These functions can be used to perform tasks like image filtering, resizing, cropping, color space conversions, histogram equalization, and more. The Python syntax makes it easy to apply these operations to images.

- **Drawing and Annotation**:

OpenCV's Python interface allows you to draw geometric shapes, text, and overlays on images. Functions like cv2.line(), cv2.rectangle(), cv2.circle(), and cv2.putText() enable you to annotate images with visual elements.

- **Object Detection and Tracking**:

OpenCV's Python bindings support various object detection and tracking algorithms. You can utilize pre-trained models or implement custom algorithms for tasks like face detection, pedestrian detection, and object tracking. OpenCV provides functions like cv2.CascadeClassifier() for Haar cascades and cv2.Tracker() for object tracking.

- **Integration with NumPy**:

OpenCV's Python interface seamlessly integrates with the NumPy library. Images loaded with OpenCV are represented as NumPy arrays, allowing for easy interoperability between OpenCV and other Python libraries for numerical computations and data manipulation.

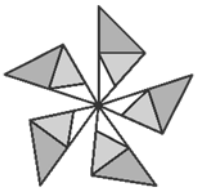
OpenCV with Python is widely used in a variety of applications, including computer vision research, robotics, video processing, augmented reality, and

more. Its Python interface makes it accessible to developers, researchers, and hobbyists who prefer Python as their primary programming language.

The combination of OpenCV's powerful computer vision capabilities and Python's simplicity and flexibility makes OpenCV with Python a popular choice for a wide range of computer vision tasks.

3.1.6 ONNX Runtime (version 1.14.0)

Overview



ONNX Runtime is an open-source runtime engine that is specifically designed for neural network inference. It provides a high-performance and efficient solution for deploying and executing machine learning models in the ONNX format.

Features

- **Model Compatibility**:

ONNX Runtime supports models that comply with the ONNX specification. The ONNX format allows you to represent machine learning models trained in popular frameworks, such as TensorFlow, PyTorch, and Caffe. By converting these models to ONNX, you can leverage ONNX Runtime to execute them without the need for framework-specific inference engines.

- **Performance Optimization**:

ONNX Runtime incorporates various optimizations to maximize the inference speed and efficiency of neural network models. It utilizes hardware

acceleration, such as GPUs and specialized accelerators, to accelerate computations. Additionally, ONNX Runtime applies graph optimizations, operator fusion, and kernel optimizations to minimize latency and improve inference performance.

- **Cross-Platform Support**:

ONNX Runtime is designed to be cross-platform, allowing you to deploy models on a wide range of hardware platforms and operating systems. It supports CPUs, GPUs, and specialized accelerators, making it suitable for deployment in cloud servers, edge devices, and IoT devices. ONNX Runtime's cross-platform compatibility ensures that models can be executed consistently across different environments.

- **Custom Operator Support**:

ONNX Runtime allows for the registration and execution of custom operators, enabling you to extend its capabilities to support custom layers or operations. This feature is beneficial when working with models that contain specialized or domain-specific operators that are not natively supported by ONNX Runtime.

- **Interoperability with Frameworks**:

ONNX Runtime seamlessly integrates with popular deep learning frameworks, including TensorFlow and PyTorch. This integration allows you to convert models from these frameworks to the ONNX format and execute them using ONNX Runtime. It promotes interoperability and facilitates the deployment of models trained in different frameworks.

3.1.7 Ultralytics YOLOv8 Algorithm

Overview

YOLO stands for "You Only Look Once," which means that instead of scanning an image multiple times to detect objects, YOLOv8 processes the entire image in one pass. It is a cutting-edge, state-of-the-art (SOTA) model that builds upon the success of previous YOLO versions and introduces new features and improvements to further boost performance and flexibility. YOLOv8 uses a deep neural network architecture called Darknet as its backbone, which consists of multiple convolutional layers. These layers are responsible for extracting features from the input image and identifying objects based on those features.

The YOLOv8 algorithm divides the input image into a grid and assigns each grid cell the responsibility of detecting objects. For each grid cell, YOLOv8 predicts bounding boxes that outline the objects, along with the class probabilities for those objects. It also predicts the confidence scores, which represent the model's confidence in the accuracy of its predictions.

One of the key advantages of YOLOv8 is its real-time processing capability. It can detect objects in images or video frames at impressive speeds, making it suitable for applications such as surveillance, autonomous vehicles, and robotics. YOLOv8 achieves this speed by utilizing techniques like anchor boxes, which help improve the accuracy of bounding box predictions, and feature pyramid networks, which enable the detection of objects at various scales.

Ultralytics, the organization behind YOLOv8, provides an open-source implementation of the algorithm, making it accessible to developers and researchers who want to utilize its capabilities for their own projects.

3.1.8 Microsoft Azure

Overview

Microsoft Azure is a cloud computing platform and set of services offered by Microsoft. It provides a range of tools and services that enable organizations to build, deploy, and manage applications and services on Microsoft-managed data centers globally.

Here's an overview of Microsoft Azure:

- Cloud Services: Azure offers a comprehensive set of cloud services, including computing, storage, networking, databases, analytics, artificial intelligence (AI), Internet of Things (IoT), and more. These services can be utilized to develop and deploy a wide range of applications and solutions.
- Scalability and Flexibility: Azure allows organizations to scale their applications and services dynamically to meet changing demands. It offers a pay-as-you-go model, enabling businesses to only pay for the resources they consume, making it cost-effective and flexible.
- Global Presence: Azure operates in numerous data centers located worldwide, providing global coverage and allowing organizations to deploy applications and services closer to their end-users, improving performance and reducing latency.
- Hybrid Capabilities: Azure offers hybrid cloud capabilities, enabling organizations to seamlessly integrate their on-premises infrastructure with Azure services. This allows for a hybrid cloud approach, providing flexibility, data synchronization, and workload mobility.

- Security and Compliance: Azure prioritizes security and compliance, offering robust measures to protect customer data. It adheres to various industry certifications and standards, ensuring compliance with regulations and guidelines.
- AI and Machine Learning: Azure provides a range of AI and machine learning services, such as Azure Cognitive Services, Azure Machine Learning, and Azure Bot Services. These services allow organizations to incorporate AI capabilities into their applications and extract insights from data.
- Serverless Computing: Azure offers serverless computing through services like Azure Functions and Logic Apps. This allows developers to focus on writing code without managing the underlying infrastructure, enabling greater agility and efficiency.

Microsoft Azure provides a comprehensive and robust cloud platform that empowers organizations to leverage a wide array of services, tools, and capabilities for building, deploying, and managing their applications and services in the cloud.

3.1.9 Flutter (version >=3.10.0)

Overview

Flutter is an open-source UI framework developed by Google that allows developers to create high-performance, cross-platform applications for mobile, web, and desktop using a single codebase. It uses a reactive programming style, making it easier to build beautiful and fluid user interfaces**. Here are some of the advantages of using Flutter:**

- **Single codebase:**

Flutter allows developers to write code once and use it across multiple platforms. This means you can develop for iOS and Android simultaneously, saving time and effort. Flutter also provides a consistent UI experience across platforms, ensuring your app looks and behaves the same way everywhere.

- **Fast development:**

Flutter's hot reload feature allows developers to see the changes they make to the code in real-time, without needing to restart the app. This greatly speeds up the development process and enables developers to experiment, iterate, and fix bugs quickly.

- **Rich widget library:**

Flutter has an extensive set of pre-designed widgets that cover a wide range of UI elements, such as buttons, lists, navigation, input fields, and more. These widgets are highly customizable, and you can create your own custom widgets to match your app's unique design requirements.

- **Expressive and beautiful UI:**

With Flutter, you have full control over every pixel on the screen. You can create stunning and expressive UI designs that closely follow your app's branding guidelines.

- **Access to native features:**

Flutter provides excellent integration with the device's native features and APIs. It offers a rich set of plugins that allow developers to access platform-specific functionality, such as camera, geolocation, sensors, storage, and more.

- **Strong developer community:**

Flutter has a vibrant and rapidly growing developer community. This means you can find a wealth of resources, tutorials, and packages to help you overcome challenges and accelerate your development process. The community actively contributes to the framework's improvement and provides valuable support.

- **Cost-effective development:**

Flutter's ability to build cross-platform apps with a single codebase significantly reduces the development and maintenance costs. It allows you to leverage the same team.

- **Hot Reload:**

Flutter's hot reload feature allows developers to instantly see the changes they make to the code reflected in the app's UI. This significantly speeds up the development process, as developers can iterate quickly, fix bugs, and experiment with different UI designs in real-time.

3.2.1 Programming language: Dart language (version >=3.0.0 <4.0.0)

Overview

Dart is a programming language developed by Google, primarily used for building applications with Flutter, although it can be used for other purposes as well. Dart combines a modern syntax with powerful features, making it a versatile language for various development scenarios. Here are some advantages of using Dart:

- **Concise and readable syntax:**

Dart has a clean and readable syntax that resembles languages like Java or JavaScript, making it easy to learn for developers familiar with these languages.

- **Optional static typing:**

Dart supports both static typing and dynamic typing. Static typing helps catch errors at compile-time, improves code quality, and enables better tooling and code analysis. However, Dart also allows dynamic typing, giving developers flexibility when needed.

- **Just-in-Time (JIT) and Ahead-of-Time (AOT) compilation**:

Dart offers the flexibility of using both JIT and AOT compilation. During development and debugging, Dart uses JIT compilation to enable features like hot reload, allowing developers to see code changes immediately.

- **Asynchronous programming:**

Dart has built-in support for asynchronous programming, making it easier to write efficient and responsive applications. It provides features like async/await and Futures for handling asynchronous operations, such as network requests or file I/O, in a concise and readable manner.

- **Object-oriented programming:**

Dart is a fully object-oriented language, supporting features like classes, inheritance, interfaces, mixins, and more. It promotes the principles of encapsulation, polymorphism, and code reusability, enabling developers to write modular and maintainable code.

- **Strong type system:**

Dart has a strong type system that enables better code analysis, early error detection, and improved tooling support. It helps catch type-related errors before runtime, reducing the chances of unexpected crashes and improving code reliability.

Packages:

- cupertino_icons: ^1.0.5

  - Provides the Cupertino icon set for use in Flutter applications.

- conditional_builder_null_safety: ^0.0.6

  - Offers conditional rendering of UI components based on null safety in Dart.

- bloc: ^8.1.0

  - Implements the Business Logic Component pattern for state management in Flutter.

- flutter_bloc: ^8.1.1

  - Integrates the bloc package with Flutter, providing additional tools and features for state management.

- dio: ^4.0.6

  - A powerful HTTP client for Dart that simplifies API communication.

- fluttertoast: ^8.1.2

  - Allows displaying toast notifications in Flutter applications.

- shared_preferences:

  - Provides a simple way to persist key-value data in Flutter using the device's local storage.
- firebase_core: ^2.4.1

  - A Flutter plugin that provides Firebase Core functionality for initializing Firebase services.
- firebase_auth: ^4.2.5

  - Enables authentication and user management using Firebase Authentication.
- cloud_firestore: ^4.3.1

  - Integrates Firestore, a NoSQL cloud database, for data storage and real-time synchronization.
- image_picker: ^0.8.6+1

  - Allows users to select images from the device's gallery or capture new ones using the camera.
- path: ^1.8.2

  - Provides utility functions for manipulating file and directory paths.
- share_plus: ^6.3.1

  - Enables sharing content from Flutter applications to other platforms using the device's share functionality.
- file_picker: ^5.2.5

  - Allows users to select files from the device's storage using a file picker dialog.

- avatar_glow: ^2.0.2

  - Creates a glowing effect around user avatars to enhance visual aesthetics.

- speech_to_text: ^6.1.1

  - Enables speech recognition capabilities, converting spoken words into text.

- http: ^0.13.6

  - Provides HTTP client functionality for making network requests in Flutter.

- chat_gpt_sdk: ^2.0.0

  - Integrates the ChatGPT SDK for building conversational AI chatbots in Flutter applications.

- flutter_tts: ^3.7.0

  - Allows text-to-speech synthesis, converting written text into spoken words.

- google_fonts: ^4.0.4

  - Offers a wide variety of beautiful fonts from the Google Fonts collection for use in Flutter applications.

- videosdk: ^1.1.3

  - Provides video communication and collaboration capabilities for integrating video functionalities into Flutter apps.

- flutter_vlc_player: ^7.2.0

- Implements a VLC-powered video player for playing various video formats in Flutter applications.

- google_ml_kit: ^0.7.3

- Integrates Google's ML Kit, offering a wide range of machine learning features, such as text recognition and image labeling.

- flutter_localizations:

- Flutter SDK package that provides localization support for internationalizing Flutter applications.

- localization: ^2.1.0

- Implements localization functionality, allowing apps to support multiple languages and adapt to different locales.

- google_maps_flutter: ^2.3.1

- Integrates Google Maps functionality into Flutter applications, enabling map display and location-related features.

- geolocator: ^7.6.0

- Provides geolocation services, allowing apps to retrieve the device's current location.

3.2.2 Firebase

Overview

It is a comprehensive mobile and web application development platform provided by Google. It offers a wide range of services and tools that developers can use to build high-quality, scalable, and feature-rich applications. Key Features and Services of Firebase:

- **Real-time Database:**

  Firebase provides a NoSQL cloud-hosted database that enables real-time synchronization of data across clients and devices. It allows developers to build collaborative and responsive applications with real-time updates.

- **Authentication**:

  Firebase offers a robust authentication system that supports various authentication methods like email/password, social media logins (Google, Facebook, Twitter), and more. It simplifies user authentication and authorization in your application.

- **Cloud Firestore:**

  Firestore is a flexible and scalable NoSQL document database provided by Firebase. It offers powerful querying, real-time synchronization, offline support, and automatic scaling to handle large amounts of data.

- **Cloud Storage:**

  Firebase Storage provides secure and reliable cloud storage for storing user-generated content such as images, videos, and files. It offers easy integration with other Firebase services and provides efficient file uploads and downloads.

- **Cloud Functions:**

  Firebase Cloud Functions allows you to run server-side code in response to events triggered by Firebase features or HTTPS requests. It enables you to extend the functionality of your app without managing infrastructure.

- **Analytics:**

  Firebase Analytics helps you gain insights into user behavior and app usage. It provides detailed analytics reports, user demographics, and event tracking, allowing you to make data-driven decisions.

- **Performance Monitoring:**

Firebase Performance Monitoring helps you monitor and optimize your app's performance. It provides detailed performance metrics, traces, and network insights to identify and fix performance bottlenecks.

- **Test Lab:**

  Firebase Test Lab provides a cloud-based infrastructure for testing your app on a wide range of devices and configurations. It helps ensure app quality and compatibility across different platforms.

- Below an image from our Cloud Firestore.

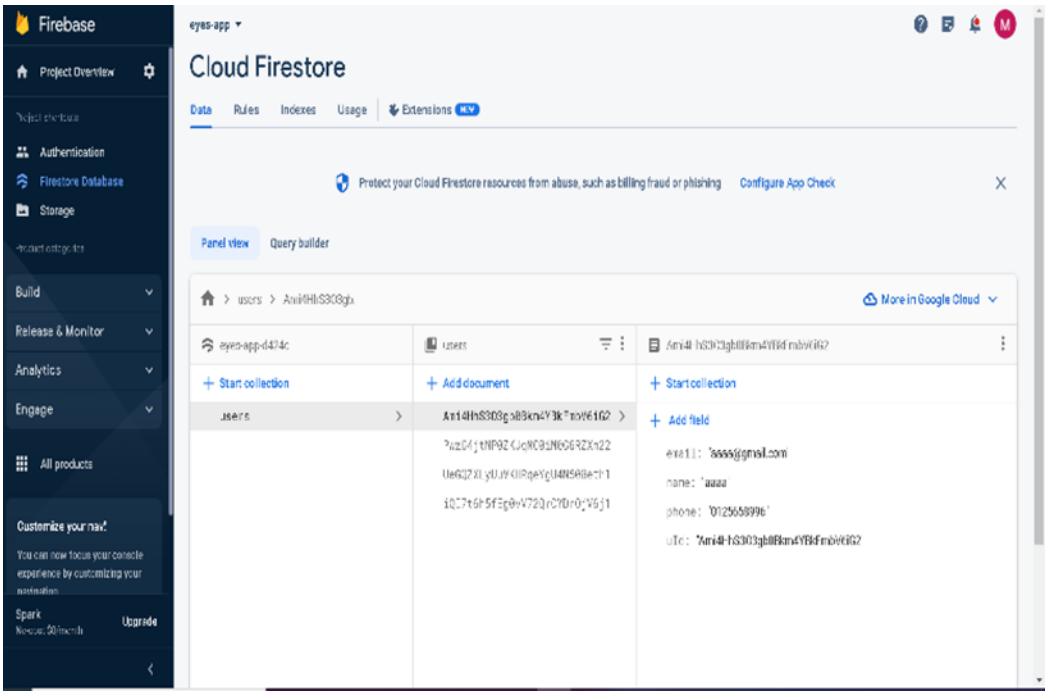


Figure 1

3.2.3 Cubit (State Management System)

Overview

First of all what is state management used for? it allows you to transfer data within the application. And whenever data is passed, the application’s state is updated, consequently rebuilding the system, ensuring that the UI reflects

the current data and reacts to user interactions appropriately. Flutter traditionally provides Stateful Widget to manage states in applications. However, we need to deal with a few limitations when using Stateful Widgets like:

- Performance Overhead: Stateful Widgets rebuild their entire subtree whenever there is a state change. This can lead to unnecessary rebuilds of unrelated widgets, affecting the performance of the application, especially in scenarios where there are frequent state changes.
- Testing Complexity: Testing Stateful Widgets can be more complex compared to testing pure functions or immutable state. The mutable nature of state in Stateful Widgets can make it harder to write comprehensive and predictable unit tests.
- Repetitive Code: Stateful Widgets require writing additional code to handle state changes. This can lead to repetitive code, especially when dealing with complex state management scenarios. This can make the codebase harder to read, and more prone to bugs and maintenance issues.

To overcome the limitations, we can choose Flutter state management using **Cubit**.

Cubit is a state management solution introduced by the Flutter team as part of the Flutter Bloc library. It provides a simple and predictable way to manage the state in Flutter applications. Cubit follows the unidirectional

data flow pattern, where the UI triggers events that are processed by the Cubit, resulting in state changes that are then reflected back in the UI. Here are some key features and concepts of Cubit:

- State: The state represents the current condition or data of the application. In Cubit, the state is steady, and any changes result in the creation of a new state object.
- Events: Events are triggers or actions that occur in the UI and are processed by the Cubit. Events represent user interactions, data updates, or any other actions that lead to state changes.
- Cubit: A Cubit is a class responsible for managing a specific piece of state and handling events related to that state. It takes an initial state and exposes methods to handle events and emit new states.
- Business Logic: Cubit encapsulates the business logic of the application. It determines how events are processed and how the state is updated based on those events. Business logic is defined within the Cubit class.
- State Changes: When an event is triggered, the Cubit processes the event and emits a new state. The state change is propagated to the UI, which updates accordingly to reflect the new state.
- Testing: Cubit provides testability by separating the business logic from the UI. It allows easy testing of the Cubit's behavior and state changes by simulating events and verifying the resulting states.

3.2.4 Google Maps Platform

Overview

The Google Maps Platform is a comprehensive suite of tools, APIs, and SDKs that enable developers to integrate maps, location-based services, and

geospatial data into their applications and websites. It provides a range of functionalities to enhance user experiences, visualize data geographically, and offer location-based services.

The platform offers various APIs and SDKs, including:

1. Maps JavaScript API: Allows developers to embed interactive maps into web applications and customize them with markers, overlays, and other visual elements.

2. Maps SDK for Android and iOS: Enables developers to integrate Google Maps directly into mobile applications, providing features like map display, geolocation, and navigation.

3. Places API: Provides access to a vast database of places, allowing developers to search for and retrieve information about businesses, points of interest, addresses, and more.

4. Geocoding API: Converts addresses into geographic coordinates (latitude and longitude) and vice versa, enabling geolocation and mapping functionalities.

5. Directions API: Offers routing and directions between multiple points on a map, including driving, walking, and transit directions.

6. Distance Matrix API: Calculates distances and travel times between multiple origins and destinations, useful for applications involving logistics, delivery services, or route optimization.

7. Street View Static API: Allows developers to embed static street view imagery into applications, offering panoramic views of locations around the world.

8. Elevation API: Provides elevation data for any location on Earth, useful for applications that require terrain analysis or altitude-based calculations.

The Google Maps Platform provides robust and reliable geospatial services, backed by Google's extensive map data and infrastructure. It offers various pricing plans and usage limits based on the specific APIs and services used.

Overall, the Google Maps Platform empowers developers to create location-aware applications with rich mapping functionalities, delivering enhanced user experiences and enabling a wide range of geospatial applications across industries such as transportation, retail, tourism, and more.

3.2.5 Android Studio

Overview

It is an Integrated Development Environment (IDE) specifically designed for Android app development. It provides developers with a comprehensive set of tools and features to build, test, and debug Android applications efficiently.

- **Development Environment:** Android Studio provides a user-friendly and feature-rich development environment for building Android apps. It offers a visually appealing and customizable interface, making it easy to navigate and work with project files.
- **Code Editor:** Android Studio includes a powerful code editor with advanced features such as syntax highlighting, code completion, code refactoring, and intelligent code analysis. It supports multiple programming languages.
- **Layout Editor:** Android Studio features a drag-and-drop layout editor that allows developers to visually design app layouts using XML or by arranging UI components.
- **Build and Compilation:** Android Studio offers robust build and compilation tools to generate APK (Android Package) files for testing and deployment. It handles resource management, compiles source code, and resolves dependencies to create a production-ready app package.
- **Emulator and Device Testing:** Android Studio comes with a built-in emulator that enables developers to test their apps on virtual Android devices. It supports various device configurations, screen sizes, and Android versions, allowing thorough testing of app behavior.

Additionally, developers can connect physical Android devices for testing directly from Android Studio.

- **Integration with Android SDK and Libraries:** Android Studio seamlessly integrates with the Android Software Development Kit (SDK) and allows easy access to Android APIs, libraries, and tools.
- **Version Control and Collaboration:** Android Studio supports popular version control systems like Git, enabling developers to manage source code repositories.

3.2.5 Screen Design Requirements

- **Figma:** is a collaborative cloud-based design tool that enables real-time design and prototyping workflows, making it ideal for remote teams. . Here are some additional details about Figma's features and benefits:
  - **Cloud-Based Design:** Figma operates entirely in the cloud, eliminating the need for local installations and ensuring that the latest version of the design is always accessible. Designers can access their projects from any device with an internet connection, enabling them to work flexibly from different locations.
  - **Version History and Collaboration History:** Figma maintains a detailed version history of design files, allowing designers to review and revert to previous versions if needed. Additionally, Figma keeps a collaboration history log, enabling users to track changes, comments, and discussions made during the design process. This promotes transparency and facilitates effective communication within design teams.
- **Adobe XD:**

Adobe XD is a powerful design and prototyping tool specifically designed for UX/UI designers, offering features for creating interactive designs and seamless integration with other Adobe Creative Cloud apps. Here are some additional details about Adobe XD and its features:

  - **Interactive Design and Prototyping**: Adobe XD allows designers to create interactive and dynamic designs with ease. Designers can define interactive elements, transitions, and animations to simulate user interactions and create a realistic user experience. This helps stakeholders and clients visualize the functionality and flow of the final product.
- **Adobe photoshop:**

Adobe Photoshop is a renowned graphic design software for logo creation. With its powerful tools, designers can create and customize logos with precision. It offers extensive editing options and versatility, making it an ideal choice for professional application logo design. Mainly used here for logo of the application.

3.3. Hardware Requirements

3.3.1. Raspberry pi 4B

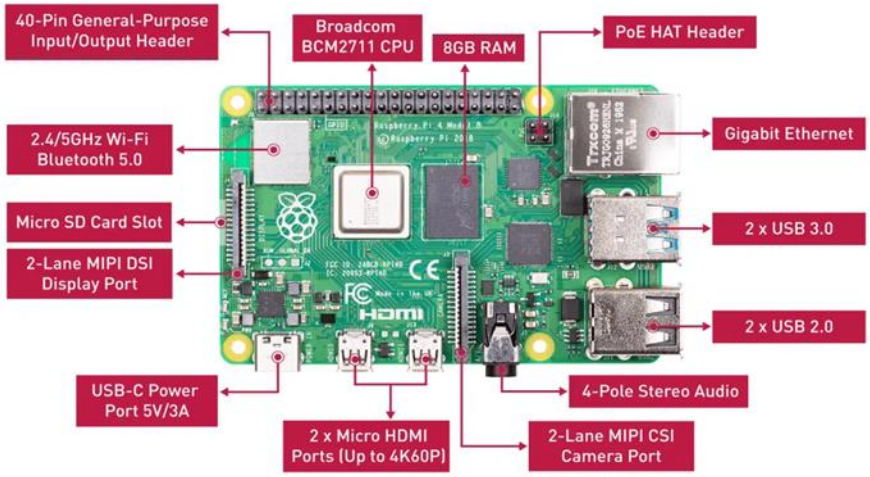
The Raspberry Pi 4 is a single-board computer developed by the Raspberry Pi Foundation. It is the latest iteration in the Raspberry Pi series, offering enhanced performance and a wide range of features compared to its predecessors. This technical report provides an overview of the Raspberry Pi 4, including its specifications, architecture, connectivity options, and applications.

Specifications

- The Raspberry Pi 4 is a notable improvement over previous models, offering substantial hardware upgrades.
- It is equipped with a Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit system-on-a-chip (SoC).
- The CPU operates at clock speeds up to 1.5 GHz, providing enhanced processing power.
- The Raspberry Pi 4 is available in three memory configurations: 1GB, 2GB, and 4GB LPDDR4-3200 SDRAM.
- The increased memory capacity allows for improved multitasking and handling of more resource-intensive applications.
- The Raspberry Pi 4 features a Video Core VI GPU, which supports OpenGL ES 3.0 graphics.
- The GPU enhancement enables better graphics rendering and improved performance in graphical applications and games.

Architecture

- The Raspberry Pi 4 retains the credit card-sized form factor, featuring a Broadcom system-on-a-chip with various components integrated onto a single board.
- It includes:
- MicroSD card slot for primary storage.
- Micro-HDMI port supporting up to two 4K displays,
- Gigabit Ethernet port, two USB 3.0 ports,
- Two USB 2.0 ports, a USB Type-C power supply port,
- A 3.5mm audio jack
- Dual-band 802.11ac Wi-Fi
- Bluetooth 5.0 connectivity.



Software Support

The Raspberry Pi 4 is compatible with various operating systems, including the official Raspberry Pi OS (formerly Raspbian), which is based on Debian Linux. It also supports other Linux distributions like Ubuntu, Fedora, and Arch Linux.

3.3.2 Raspberry Pi Camera Module V2

The Raspberry Pi Camera Module V2 is the second-generation camera module designed specifically for use with Raspberry Pi boards. It is a small, high-definition camera that connects to the Raspberry Pi via the CSI (Camera Serial Interface) port.

3.3.3 Microphone

To connect a microphone to a Raspberry Pi 4, you'll typically need a USB or 3.5mm (TRS or TRRS) microphone.

USB microphone: Connect to USB port and configure audio input settings.

3.5mm microphone: Connect to audio jack and configure audio input settings.

In this project we use usb mic.

3.3.4 Headphone

The Raspberry Pi 4 can be used with headphones as an audio output. The Raspberry Pi 4 has a 3.5mm audio jack that can be used to connect headphones or speakers.

3.3.5 Touch sensor

Touch sensors are input devices that detect physical contact or touch. They are widely used in applications such as smartphones, tablets,and interactive displays. Capacitive touch sensors are popular due to their accuracy, responsiveness, and durability. These sensors operate by measuring changes in capacitance caused by the presence of a conductive object, such as a human finger.

Chapter 4
Implementation

4.1 Artificial Intelligence & Computer Vision

Face recognition

Face recognition is a technology that involves automatically identifying or verifying individuals by analyzing and comparing their facial features from images or video data. It is a subset of biometric identification systems that utilize unique characteristics of a person's face for identification purposes.

The process of face recognition typically involves the following steps:

- **Face Detection:**

  There are various ways to implement face detection yet the most accurate is to use a CNN object detector model trained to detect faces.

  the model we used is the **yunet model** which is a lightweight, accurate, and robust face detector. It's also integrated into OpenCV as (**cv2.FaceDetectorYN**) making it easy to use.

- **Feature Extraction:**

  Each detected face is fed to another CNN network that is used to extract features from the face; the features are actually just a 192-d vector that contains values that serve as the encodings or the embeddings of the face.

The model used for this purpose is the MobileFaceNet that is a lightweight yet effective model at encoding faces. The model is trained to minimize the distance between faces of the same person even if they're in different conditions and maximize the distance between faces of different persons.
The extracted encodings are then used for comparison and identification.

- **Face Identification:**

The encoded face features are compared against a pre-existing database of known faces encodings. The system uses **Euclidean Distance** algorithms to measure the similarity or dissimilarity between the features of the detected face and the reference faces in the database. If the distance between two encoding is less than a predefined threshold value (0.8 in our case) then the two faces are the same otherwise they are not.

The Face recognition system is used for Identity Verification. It is used to verify the identity of an individual to tell the user who they are.

Banknotes Recognition

Banknote recognition refers to the process of automatically identifying and classifying different types of banknotes, typically based on their visual features. It involves using computer vision techniques and machine learning algorithms to detect and recognize banknotes accurately. We classify the Egyptian banknote from 1 pound to 200 pounds.

By combining PyTorch for model training and ONNX Runtime for deployment, we can train, and deploy efficient and accurate banknote recognition systems. Image Acquisition: Banknote images are captured using the camera.

One problem we faced while developing this was that there was no public dataset containing Egyptian banknote images therefore we had to collect the data ourselves. The data was collected by simply Synthesizing images as follows:

- First we collect banknote images of high quality and different conditions using **Bing search Engine** and its **image search feature**.
- Then prepare images to serve as backgrounds for the banknotes.
- Perform some augmentation and distortion to the banknote images to have a diverse range of banknotes with different positions and conditions.
- Finally lay out the banknotes on the backgrounds.

  Below is an example of a banknote and a datapoint generated by laying out a banknote on a background.

Model training:

We used **Pytroch** for training our CNN model to classify banknotes. One way we could take is to build a CNN model from scratch and train it yet this is suboptimal

because we don't have enough data to train the model from scratch therefore we can perform the training in a different and more effective way that is **transfer learning**. Transfer learning simply means that instead of training a network from scratch on the task in hand we can find a network that is trained on a similar task and continue from there to train our network. That has many benefits like faster convergence of the model and much less data required.

The model we used is the **Efficientnet_b0** model that was pre trained on **Imagenet** which is a diverse and large scale benchmark for image classification.

the model

All the code for preparing the data and training the network is provided in the appendix.

Exporting to ONNX for Deployment:

ONNX (Open Neural Network Exchange) is an open standard for representing machine learning models, allowing interoperability across different frameworks and platforms. ONNX Runtime is a runtime engine that executes ONNX models efficiently. When developing a banknote recognition system, ONNX Runtime can be leveraged for:

- Model Export: PyTorch models can be exported to the ONNX format using the torch.onnx.export() function. This enables compatibility and portability of the trained model across platforms and frameworks.
- Inference and Deployment: ONNX Runtime provides a high-performance engine for deploying and executing the trained banknote recognition model. It ensures efficient and optimized execution, enabling real-time banknote recognition in production environments.
- Platform Compatibility: ONNX Runtime supports various platforms and devices, including CPUs, GPUs, and specialized hardware accelerators. This

allows banknote recognition models to be deployed on a wide range of devices, from edge devices to cloud servers.

The trained model is then exported to **onnx** format to be used in production using **onnxruntime.**

Object detection

Overview

Object detection is a computer vision technique that involves identifying and localizing objects within images or videos. The goal is to automatically locate and classify objects of interest within a given scene.

Object detection typically follows these steps:

- Input data: Obtain the input data, which can be images or video frames, on which you want to perform object detection.
- Preprocessing: Prepare the input data by resizing, normalizing, or applying any necessary transformations to ensure it is compatible with the object detection algorithm.
- Feature extraction: Extract features from the input data using techniques such as convolutional neural networks (CNNs). CNNs are deep learning models designed to automatically learn and represent visual features from the input images.
- Object localization: Determine the presence and location of objects in the input data. This is typically achieved by generating bounding boxes that tightly enclose the objects of interest. Some algorithms use predefined

bounding boxes, while others dynamically adjust the box coordinates based on the detected object's characteristics.

- Object classification: Assign labels or class probabilities to the detected objects. This step involves identifying the type or category of each object, such as "car," "person," or "cat." Classification can be performed using various machine learning algorithms, including deep neural networks.
- Post-processing: Refine the object detection results by removing duplicate or low-confidence detections. Techniques like non-maximum suppression are often applied to suppress multiple overlapping bounding boxes and retain only the most confident and accurate detections.
- Visualization and analysis: Finally, visualize the results by overlaying the bounding boxes and class labels on the input data. The detected objects can be analyzed, counted, tracked, or used for further downstream tasks, depending on the specific application.

To perform object detection with YOLOv8, you would typically follow these steps:

- Installation and setup: Start by installing the necessary dependencies and configuring the YOLOv8 implementation provided by Ultralytics. This may involve setting up the required deep learning frameworks like PyTorch and downloading the pre-trained YOLOv8 model weights.
- Input preparation: Prepare the input data, which could be images or video frames, to be processed by the YOLOv8 algorithm. This may involve resizing, cropping, or normalizing the input to match the required format.
- Running inference: Pass the prepared input through the YOLOv8 algorithm for object detection. YOLOv8 performs a forward pass through its neural network architecture to generate predictions for objects present in the input

data. These predictions typically include bounding box coordinates, class labels, and confidence scores for each detected object.

- Post-processing: Apply post-processing techniques to refine the object detection results. This may involve filtering out low-confidence detections, removing redundant detections using non-maximum suppression, or applying additional constraints based on specific application requirements.
- Visualization and analysis: Finally, visualize the detected objects by overlaying the bounding boxes on the original input data. You can also analyze the detection results, extract relevant information, or use them for further downstream tasks.

Ultralytics provides an open-source implementation of YOLOv8, which is widely used and actively maintained. It offers various pre-trained models trained on large datasets and provides a user-friendly interface for running object detection experiments with YOLOv8.

Color Recognition

Overview

Color detection is a computer vision task that involves identifying and recognizing colors within images or videos. The goal is to automatically determine the dominant or specific colors present in the input data.

Color detection typically follows these steps:

- Input data: Obtain the input data, which can be images or video frames, on which you want to perform color detection.
- Preprocessing: Prepare the input data by resizing, normalizing, or applying any necessary transformations to ensure consistent and accurate color analysis.

- Color space conversion: Convert the input data from its original color space (such as RGB) to a different color space that is more suitable for color analysis. Popular color spaces used for color detection include HSV (Hue, Saturation, Value) and Lab (Lab*).
- Color classification or clustering: Assign labels or group similar colors together. Depending on the specific task and requirements, color detection can be approached as either a classification problem, where predefined color categories are assigned, or as a clustering problem, where colors are grouped based on their similarity.

Color recognition can be performed in various ways. It seems like a trivial task yet it's difficult to create an accurate color recognizer because of the camera and lighting and other factors that can affect the accuracy.
Color recognition is implemented as follows:

- Create a database containing multiple RGB values for different colors to have a diverse range covering each color.
- Import libraries: Begin by importing the necessary libraries, including OpenCV and NumPy.
- Capture an image and extract the center 50 x 50 pixels of the image.
- average the extracted values.
- find the distance between the value and all the values in the colors data using **Euclidean Distance** algorithms to find the nearest candidate to this color.

Optical Character Recognition

Overview

OCR (Optical Character Recognition) is a technology that allows computers to extract text from images or scanned documents and convert it into machine-readable text. OCR enables automated text recognition, making it possible to process and analyze text data that is present in non-editable formats.

OCR engine:

The Read OCR engine from Microsoft is made up of numerous sophisticated machine-learning models that support various languages. It can extract text from both printed and handwritten documents, even when the languages and writing styles are intermingled. For flexible deployment, Read is accessible as a cloud service and an on-premises container. The most recent preview also includes speed improvements that make it simpler to develop OCR-assisted user experiences as a synchronous API for single, non-document, image-only scenarios.

How OCR works in general

Utilizing a scanner to process a document's physical shape is the initial phase of OCR. OCR software turns the document into a two-color, or black and white, version after all pages have been copied. Dark areas are determined to be characters that need to be recognized, and bright parts are determined to be background, in the scanned-in image or bitmap.

Then, additional processing is done on the dark areas to uncover alphabetic or numeric digits. Although the methods used by OCR applications can differ, they

typically focus on one character, word, or block of text at a time. Then, one of two algorithms is used to identify the characters:

1- Pattern recognition. OCR software compares and recognizes characters in the scanned document using examples of text in different fonts and formats that are fed into the computer.

2- Feature detection. To identify characters in the scanned document, OCR software employs rules pertaining to the characteristics of a particular letter or number. The number of angled lines, crossing lines, or curves in a character could constitute characteristics for comparison. For instance, the capital "A" may be encoded as two diagonal lines that come together in the middle and form a line.

When a character is identified, it is converted into an ASCII code that can be used by computer systems.

Our implementation to OCR

We use python and Microsoft azure cognitive service.

necessary packages

1- os module: This module offers a portable method of using functionality that is dependent on the operating system. Open() can be used to simply read or write a file, the os.path module can be used to change paths, and the fileinput module can be used to read every line in every command-line file. The tempfile module should be used to create temporary files and directories, and the shutil module should be used to handle high-level file and directory operations.

2- io module: The primary tools available in Python for handling different I/O types are found in the io module. Text I/O, binary I/O, and raw I/O are the three basic categories of I/O. Each of these can be utilized with different backing stores

because they are general categories. A file object is a physical item that fits into one of these categories. The phrases stream and file-like entity are also frequently used. Each concrete stream object, regardless of category, will have a range of capabilities: it may be read-only, write-only, or read-write. Additionally, it may provide just sequential access (as in the case of a socket or pipe) or arbitrary random access (searching forwards or backwards to any position).

3- json package: built-in package for encoding and decoding JSON data.

4- time module: The time module in Python provides various functions to work with time-related tasks. It allows you to access the system time and also manipulate time values.

5- gTTS library: gTTS, or "Google Text-to-Speech," is a Python library that allows you to convert text to speech using the Google Text-to-Speech API.

6- ComputerVisionClient class: Modern algorithms are made available by the Computer Vision API to process photos and produce data. It can be used to find all the faces in an image, for instance, or to evaluate whether an image contains mature content. Other functions include identifying the content of photographs, predicting dominant and accent colors, and describing an image with full English words. To display huge photos properly, it can also intelligently create image thumbnails.

7- OperationStatusCodes module: In the Azure Cognitive Services Computer Vision API, OperationStatusCodes is a module that defines the various status codes that indicate the status of an operation, such as image analysis or text recognition. These codes provide information on whether the operation is in progress, completed successfully, or encountered an error.

8- msrest.authentication module: CognitiveServicesCredentials class from the msrest.authentication module. In the Microsoft REST (Representational State Transfer) library, the CognitiveServicesCredentials class is used for authenticating requests to Microsoft Cognitive Services APIs, such as Computer Vision, Speech, and Language Understanding.

9- PIL module: The PIL module is a popular library for handling and manipulating images in Python.

By importing the Image class, you can create and work with image objects, perform operations like resizing, cropping, and rotating images, and apply various filters and enhancements.

The ImageDraw class allows you to draw shapes, lines, and text on your image. It provides functions to create and manipulate image drawings, such as rectangles, circles, and polygons.

The ImageFont class is used for loading and rendering fonts on images. It allows you to specify the font type, size, and style to apply when adding text to your image.

10- langdetect library: is a Python package that enables language detection from text. It provides a simple and convenient way to identify the language of a given piece of text.

OCR functionality is provided as part of the Azure Cognitive Services suite offered by Microsoft. The Azure Cognitive Services OCR API allows you to integrate OCR capabilities into your applications using the power of Microsoft's cloud infrastructure. Here's an overview of how to use OCR with Microsoft Azure:

- Create an Azure Cognitive Services resource: Start by creating an Azure account if you don't have one. Then, create a Cognitive Services resource

in the Azure portal. This resource will provide the necessary API key and endpoint for accessing the OCR service.

- Get API key and endpoint: Once you have created the Cognitive Services resource, obtain the API key and endpoint from the Azure portal. These credentials will be used to authenticate and access the OCR service.
- Make API requests: Use an HTTP client or SDK to make requests to the OCR API. Construct an HTTP POST request to the OCR endpoint, providing the image data or the URL of the image you want to perform OCR on. Include the API key as a header in the request for authentication.
- Process the OCR response: Receive the response from the OCR API, which will contain the recognized text extracted from the image. The response may also include information such as word bounding boxes, confidence scores, and language identification. Extract the relevant information from the response and use it as needed in your application.

Microsoft Azure OCR supports various languages and provides features such as language detection, text layout analysis, and image preprocessing for improved OCR accuracy. Additionally, Azure Cognitive Services offer SDKs for various programming languages, making it easier to integrate OCR functionality into your applications.

It's important to review the pricing details and any usage limits associated with the Azure Cognitive Services OCR API to understand the cost implications and ensure compliance with your specific requirements.

4.2 Hardware

1 Raspberry pi 4

I. Introduction

The Raspberry Pi 4 is a single-board computer developed by the Raspberry Pi Foundation. It is the latest iteration in the Raspberry Pi series, offering enhanced performance and a wide range of features compared to its predecessors. This technical report provides an overview of the Raspberry Pi 4, including its specifications, architecture, connectivity options, and applications.

II. Connectivity Options

The Raspberry Pi 4 offers versatile connectivity options, making it suitable for a wide range of applications. It features the USB ports for connecting peripherals, such as keyboards, mice, and external storage devices. The micro-HDMI ports enable high-resolution display connections. The Gigabit Ethernet port facilitates wired network connectivity, while the built-in Wi-Fi and Bluetooth modules allow for wireless communication. The microSD card slot serves as the primary storage option, but it also supports booting from an external USB device.

III. Software Support

The Raspberry Pi 4 is compatible with various operating systems, including the official Raspberry Pi OS (formerly Raspbian), which is based on Debian Linux. It also supports other Linux distributions like Ubuntu, Fedora, and Arch Linux. Furthermore, alternative operating systems like Windows 10 IoT Core and various media center distributions are available for specific use cases.

IV. Performance and Applications

The improved specifications of the Raspberry Pi 4 contribute to enhanced performance and expanded application possibilities. With its faster processor, increased memory, and improved graphics capabilities, the Raspberry Pi 4 can handle more demanding tasks, such as multimedia processing, gaming, and IoT applications. It can serve as a versatile desktop computer, a media center, a home server, a development platform, and a learning tool for programming and electronics.

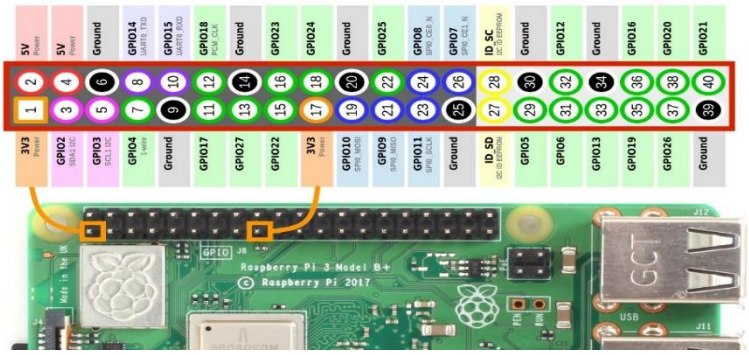
2 Input

2.1 GPIO

The General-Purpose Input/Output (GPIO) pins on the Raspberry Pi 4 are essential for interfacing the board with external devices.

I. GPIO Pin Configuration

The Raspberry Pi 4 features a total of 40 GPIO pins, numbered from GPIO2 to GPIO27, with additional power (3.3V and 5V), ground (GND), and other auxiliary pins. The GPIO pins are accessed via the 40-pin header located on the board. These pins can be configured as inputs or outputs to communicate with a wide range of devices, including sensors, actuators, LEDs, motors, and more.



II. Electrical Characteristics

The GPIO pins on the Raspberry Pi 4 operate at 3.3 volts, making them compatible with most common logic level devices. However, it is important to note that these pins are not 5V tolerant. Therefore, using voltages greater than 3.3V risks damaging the Raspberry Pi. To interface with devices operating at 5V or higher voltages, level shifters or voltage dividers are required.

III. Programming Interfaces

The Raspberry Pi 4 offers several programming interfaces to interact with the GPIO pins. The most used programming languages include Python, C, C++, and JavaScript. Libraries and modules, such as RPi.GPIO for Python, WiringPi for C/C++, and OnOff for JavaScript, provide convenient abstractions to control the GPIO pins. These libraries offer functions to set pin modes, read inputs, write outputs, and handle interrupts.

IV. GPIO Pin Modes

GPIO pins can be configured in three primary modes: input, output, and alternate function.

- In input mode, the pin is used to read the state of an external device or sensor.
- In output mode, the pin can be set to high (3.3V) or low (0V) to control the behavior of an external device.
- The alternate function mode allows GPIO pins to perform additional functionalities, such as UART, I2C, SPI, PWM, and more.

V. Interfacing with Sensors and Actuators

The GPIO pins on the Raspberry Pi 4 enable seamless integration with various sensors and actuators. For example, to connect a digital sensor, such as a motion sensor, one GPIO pin is used as an input to receive the sensor's signal. To interface with an actuator, such as an LED or motor, a GPIO pin is configured as an output to control the actuator's state.

VI. Additional GPIO Features

The Raspberry Pi 4 GPIO pins offer additional features and capabilities, enhancing their versatility and functionality. Some notable features include:

- Pull-up and pull-down resistors: These resistors can be enabled or disabled on GPIO pins to stabilize input states when external devices are not actively driving the pin.
- PWM (Pulse Width Modulation): Several GPIO pins on the Raspberry Pi 4 support hardware PWM, allowing precise control of analog-like signals for applications like motor speed control or LED dimming.
- Interrupts: GPIO pins can be configured to generate interrupts when a specific event occurs, such as a rising or falling edge, enabling efficient event-driven programming.

VII. Practical Applications

The GPIO pins on the Raspberry Pi 4 facilitate a broad range of applications. Some common use cases include:

- Home automation: Control lights, sensors, and actuators to create a smart home environment.
- Robotics: Interface with motors, sensors, and other robotic components for building robots and automation projects.
- IoT devices: Connect and communicate with various sensors

2.2 Touch sensor

I. Touch Sensor Technology

Touch sensors are input devices that detect physical contact or touch. They are widely used in applications such as smartphones, tablets, and interactive displays. Capacitive touch sensors are popular due to their accuracy, responsiveness, and durability. These sensors operate by measuring changes in capacitance caused by the presence of a conductive object, such as a human finger.

II. Touch Sensor advantages over buttons

- Aesthetics: Sleek and modern design.
- Durability: No moving parts, longer lifespan.
- Versatile Input: Supports gestures and complex interactions.
- Larger Active Area: Enables multi-touch gestures and precise tracking.
- Responsive Feedback: Instant touch detection and feedback.
- Customizable: Programmable for specific applications.
- Seamless Integration: Easy connectivity with Raspberry Pi GPIO.
- Intuitive User Experience: Familiar and user-friendly interface.
- Space-Saving Design: Eliminates physical buttons, flexible placement.
- Multi-Functionality: Can perform multiple actions within a single touch area.

These advantages highlight the superior aesthetics, durability, versatility, responsiveness, and customization capabilities of touch sensors compared to buttons when integrated with Raspberry Pi projects.

2.3 Touch sensor Integration with GPIO

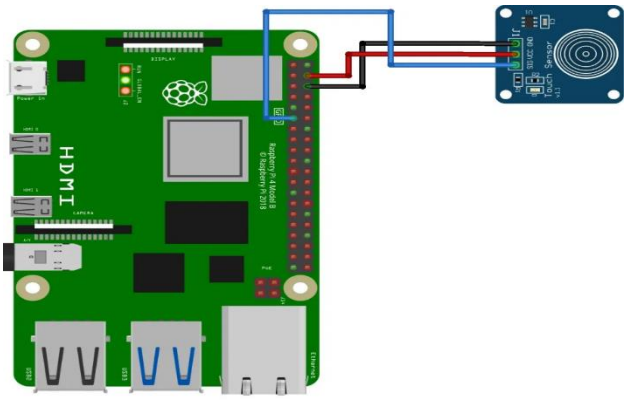
Integrating two touch sensors with GPIO on a Raspberry Pi is for the purpose of controlling software AI functions.

The objective is to enable forwarding and backwarding actions between different software functions using interrupt-based touch sensor input.

I. Hardware Setup

Steps:

- Identifying and selecting two available GPIO pins on the Raspberry Pi for connecting the touch sensors.
- Connecting the VCC (power) and GND (ground) pins of each touch sensor to the respective 3.3V and GND GPIO pins on the Raspberry Pi.
- Connecting the signal pins of the touch sensors to the chosen GPIO pins.



II. GPIO Configuration

Steps:

- Utilizing a GPIO library (such as RPi.GPIO for Python) to access GPIO functionality and interrupt support.
- Configuring the two selected GPIO pins as inputs using the library's functions or methods.
- Enabling interrupts for those GPIO pins to detect touch events using the library's interrupt handling features.

III. Programming Logic

Enable forwarding and backwarding actions using interrupts by:

- Defining separate interrupt service routines (ISRs) for the forwarding and backwarding touch sensors.

- Within each ISR, implementing the necessary code to trigger the desired software functions or switch between functions.
- Registering the ISRs for the corresponding GPIO pins and start the interrupt handling mechanism.

i. Forwarding Functionality

When the forwarding touch sensor is triggered:
The forwarding ISR is invoked, interrupting the program execution.
Within the ISR, execute the necessary code to switch to the next software function or perform the desired forwarding action.

ii. Backwarding Functionality

When the backwarding touch sensor is triggered:
The backwarding ISR is invoked, interrupting the program execution.
Within the ISR, execute the necessary code to switch to the previous software function or perform the desired backwarding action.

In conclusion, integrating touch sensors with GPIO on a Raspberry Pi using interrupts allows for the implementation of forwarding and backwarding functionality between software functions. This approach enhances user interactivity and provides a seamless and intuitive experience. The integration of touch sensors with GPIO and interrupts showcases the potential of the Raspberry Pi platform in creating interactive and responsive applications.

2.4 Microphone

I. Connecting a Microphone to a Raspberry pi 4 as input device:

To connect a microphone to a Raspberry Pi 4, you'll typically need a USB or 3.5mm (TRS or TRRS) microphone. Here are the steps to connect each type of microphone:

i. USB Microphone:

- Power off your Raspberry Pi 4 and make sure it is disconnected from any power source.
- Locate one of the available USB ports on the Raspberry Pi 4.

- Connect the USB end of the microphone to the USB port on the Raspberry Pi 4.
- Power on the Raspberry Pi 4.

Once the Raspberry Pi 4 is powered on, it should detect the USB microphone automatically. You may need to configure the audio input settings on the Raspberry Pi to use the USB microphone as the default input device.

ii. 3.5mm Microphone:

- Power off your Raspberry Pi 4 and make sure it is disconnected from any power source.
- Locate the 3.5mm audio jack on the Raspberry Pi 4. It is typically labeled with a headphone or microphone icon.
- Plug the 3.5mm TRS or TRRS connector of the microphone into the corresponding audio jack on the Raspberry Pi 4.
- Power on the Raspberry Pi 4.

Once the Raspberry Pi 4 is powered on, you may need to configure the audio input settings to use the 3.5mm audio jack as the default input device. This can be done through the Raspberry Pi's audio settings or the command-line interface.
If you're using a TRRS microphone with an integrated headphone output, you can connect headphones to the microphone's headphone jack for monitoring the audio input.
After connecting the microphone, you can use software libraries or tools, such as ALSA or PulseAudio, to capture and process the audio input on your Raspberry Pi 4. In our project we use usb mic.

II. Connecting Raspberry Pi 4 with USB Microphone for Speech Recognition

i. Introduction:

The process of connecting a USB microphone is to a Raspberry Pi 4 for speech recognition purposes as input device. The Raspberry Pi 4 is a popular single-board computer capable of running various speech recognition applications, and a USB microphone offers a convenient solution for capturing high-quality audio input.

ii. Equipment Required:

To establish the connection, the following equipment is needed:

- Raspberry Pi 4 board
- USB microphone compatible with the Raspberry Pi
- USB cable for connecting the microphone
- HDMI monitor or display (optional)
- Keyboard and mouse (optional)
- Power supply for the Raspberry Pi 4

iii. Connection Procedure:

Follow these steps to connect the USB microphone to the Raspberry Pi 4:

- Power off the Raspberry Pi 4 and ensure it is disconnected from any power source.
- Locate one of the available USB ports on the Raspberry Pi 4.
- Connect the USB end of the microphone's cable to the USB port on the Raspberry Pi 4.
- Power on the Raspberry Pi 4.
- Once the Raspberry Pi 4 is powered on, it should automatically detect the USB microphone. You may need to configure the audio input settings on the Raspberry Pi to use the USB microphone as the default input device.

iv. Software Configuration:

After connecting the USB microphone, the following software configurations are required:

- Ensure that the necessary software libraries and tools for speech recognition are installed on the Raspberry Pi 4. Popular options include CMUSphinx, Google Cloud Speech-to-Text API, or Mozilla DeepSpeech.
- Configure the audio settings on the Raspberry Pi 4 to use the USB microphone as the default input device. This can typically be done through the audio settings or preferences of the operating system or speech recognition software.
- Depending on the speech recognition library or API you choose, you may need to install additional dependencies or configure specific settings to enable speech recognition with the connected USB microphone.

v. Implementation:

With the Raspberry Pi 4 connected to the USB microphone and the necessary software configurations in place, you can proceed with the implementation of speech recognition:

- Develop or use speech recognition software or libraries compatible with the Raspberry Pi 4 and the chosen speech recognition technology.
- Write code to capture audio input from the USB microphone using the appropriate software library or API.
- Process the captured audio data for speech recognition, applying algorithms and models specific to the chosen speech recognition technology.
- Analyze the recognized speech output, perform desired actions, or extract relevant information based on the recognized speech.

vi. Conclusion:

Connecting a USB microphone to a Raspberry Pi 4 for speech recognition is a straightforward process. By following the provided steps and configuring the necessary software
settings, you can establish the connection and implement speech recognition applications on the Raspberry Pi 4. Ensure that you have the required software libraries and tools for speech recognition installed and consult the documentation or user manual provided with your USB microphone for any specific considerations.

The USB microphone offers an efficient way to capture high-quality audio input for speech recognition tasks, enabling a range of applications such as voice-controlled assistants, voice commands, and more on the Raspberry Pi 4 platform.

2.5 Raspberry Pi Camera

I. The Raspberry Pi Camera Module V2

The Raspberry Pi Camera Module V2 is the second-generation camera module designed specifically for use with Raspberry Pi boards. It is a small, high-definition camera that connects to the Raspberry Pi via the CSI (Camera Serial Interface) port. Here are some key features and specifications of the Raspberry Pi Camera V2:

- Resolution: The camera module supports a resolution of up to 8 megapixels, capturing still images and video with excellent clarity.
- Image Sensor: It uses a Sony IMX219 8-megapixel image sensor, which allows for detailed and high-quality images.
- Video Recording: The camera can record video at various resolutions, including 1080p at 30 frames per second (fps), 720p at 60 fps, and VGA (640x480) at 90 fps.
- Still Images: It can capture still images with a maximum resolution of 3280x2464 pixels.
- Lens: The camera module has a fixed-focus lens with an aperture of f/2.0. It provides a field of view (FoV) of approximately 62.2 degrees diagonally.
- Connectivity: The camera connects to the Raspberry Pi board via a ribbon cable that attaches to the CSI connector. This connector provides both data and power to the camera module.
- Compatibility: The Raspberry Pi Camera V2 is compatible with various Raspberry Pi models, including the Raspberry Pi 4 Model B, Raspberry Pi 3 Model B+, Raspberry Pi 3 Model A+, Raspberry Pi Zero, and Raspberry Pi Zero W.
- Software Support: The camera module is supported by the official Raspberry Pi operating system, Raspbian (now called Raspberry Pi OS), and various third-party software libraries, making it easy to integrate into your projects.

The Raspberry Pi Camera V2 is a versatile and affordable camera module, allowing you to capture photos and videos directly from your Raspberry Pi board. It has been widely used in projects such as surveillance systems, time-lapse photography, robotics, computer vision applications, and more.

II. Connect Raspberry pi 4 with Raspberry Pi Camera Module V2 as input device

To connect a Raspberry Pi 4 with the Raspberry Pi Camera V2, you'll need to follow these steps:

- Power off your Raspberry Pi 4 and make sure it is disconnected from any power source.
- Locate the CSI (Camera Serial Interface) connector on the Raspberry Pi 4. It is a rectangular-shaped connector near the HDMI port.
- Take the ribbon cable that comes with the Raspberry Pi Camera V2 and connect one end of the cable to the CSI connector on the Raspberry Pi 4. The metal contacts on the cable should be facing away from the HDMI port.
- Gently lift the small plastic tabs on either side of the CSI connector on the Raspberry Pi 4.

- Align the other end of the ribbon cable with the CSI connector on the Raspberry Pi Camera V2. Ensure that the metal contacts on the cable are facing towards the camera module.
- Insert the ribbon cable into the CSI connector on the camera module, making sure it is fully inserted.
- Push down the plastic tabs on either side of the CSI connector on the Raspberry Pi 4 to secure the ribbon cable in place. Ensure it is connected firmly but be careful not to apply excessive force.

Now, your Raspberry Pi 4 is connected to the Raspberry Pi Camera V2. You can power on your Raspberry Pi 4, and it should detect the camera module automatically.

After connecting the camera, you can use various software libraries, such as the picamera library in Python or the raspistill and raspivid command-line utilities, to capture images and videos with the Raspberry Pi Camera V2. Make sure you have enabled the camera interface in the Raspberry Pi's configuration settings (raspi-config) if it's not already enabled.

III. connecting a Raspberry Pi 4 with the Raspberry Pi Camera V2 for the implementation of a face recognition system

i. Introduction:

The purpose of this report is to outline the process of connecting a Raspberry Pi 4 with the Raspberry Pi Camera V2 for the implementation of a face recognition system. The Raspberry Pi Camera V2 offers high-definition image capture capabilities, making it suitable for various computer vision applications, including face recognition.

ii. Equipment Required:

To establish the connection, the following equipment is needed:

- Raspberry Pi 4 board
- Raspberry Pi Camera V2 module
- Ribbon cable (included with the camera module)
- HDMI monitor or display (optional)
- Keyboard and mouse (optional)
- Power supply for the Raspberry Pi 4

iii. Connection Procedure:

Follow these steps to connect the Raspberry Pi 4 with the Raspberry Pi Camera V2:

- Power off the Raspberry Pi 4 and ensure it is disconnected from any power source.
- Locate the CSI (Camera Serial Interface) connector on the Raspberry Pi 4. It is a rectangular-shaped connector near the HDMI port.
- Take the ribbon cable included with the Raspberry Pi Camera V2 and connect one end of the cable to the CSI connector on the Raspberry Pi 4. Ensure that the metal contacts on the cable face away from the HDMI port.
- Lift the small plastic tabs on either side of the CSI connector on the Raspberry Pi 4.
- Align the other end of the ribbon cable with the CSI connector on the Raspberry Pi Camera V2. Ensure that the metal contacts on the cable face towards the camera module.

- Insert the ribbon cable into the CSI connector on the camera module, ensuring it is fully inserted.
- Push down the plastic tabs on either side of the CSI connector on the Raspberry Pi 4 to secure the ribbon cable in place. Be cautious not to apply excessive force

iv. Software Configuration:

After connecting the camera module, the following software configurations are required:

- Power on the Raspberry Pi 4 and log in to the operating system.
- Ensure that the camera interface is enabled in the Raspberry Pi's configuration settings (raspi-config). Navigate to the interface options and enable the camera.
- Install the required software libraries for face recognition, such as OpenCV or dlib, using the package manager or by following the installation instructions provided by the respective libraries.
- Use the appropriate programming language, such as Python, to write code for face recognition using the connected camera module. Utilize the camera interface library, such as picamera, to access the camera feed and process images.

v. Implementation:

With the Raspberry Pi 4 and the Raspberry Pi Camera V2 connected, and the necessary software configurations in place, you can now implement the face recognition system. This involves utilizing the camera feed to capture images, processing them using face detection and recognition algorithms, and performing the desired actions based on the identified faces.

The implementation steps will depend on the specific face recognition library or framework being used. However, the general workflow typically involves capturing frames from the camera feed, detecting faces within the frames, extracting facial features or embeddings, comparing them with known faces, and making predictions or triggering actions based on the identified individuals.

vi. Conclusion:

Connecting a Raspberry Pi 4 with the Raspberry Pi Camera V2 for face recognition is a straightforward process. By following the provided steps, you can establish the connection and begin implementing a face

recognition system using the camera feed. Ensure that you have the necessary software libraries and programming knowledge to handle the image processing and face recognition algorithms effectively.

The Raspberry Pi Camera V2's high-definition capabilities
combined with.

IV. connecting a Raspberry Pi 4 with the Raspberry Pi Camera V2 for the implementation of a color recognition

To connect a Raspberry Pi 4 with the Raspberry Pi Camera V2 for color recognition, you'll need to follow these steps:

- Ensure your Raspberry Pi 4 is powered off and unplugged.
- Locate the CSI (Camera Serial Interface) port on the Raspberry Pi 4. It is a small, rectangular connector located near the HDMI port.
- Gently lift the plastic tab on the CSI port to reveal the metal contacts.
- Take the ribbon cable of the Raspberry Pi Camera V2 and insert it into the CSI port, with the metal contacts facing away from the
- HDMI port. Make sure it is inserted securely.
- Once the ribbon cable is inserted, push the plastic tab back down to secure the cable in place.
- Power on your Raspberry Pi 4.

Now that the hardware connection is complete, you'll need to enable the camera interface and install the necessary software libraries to work with the camera. Here are the steps:

- Boot up your Raspberry Pi 4 and open a terminal or SSH into the Pi.
- Type the following command to open the Raspberry Pi Configuration tool:

```
sudo raspi-config
```

- In the configuration tool, use the arrow keys to navigate to "Interfacing Options" and press Enter.
- Select "Camera" and press Enter.
- Choose "Yes" to enable the camera interface.
- Press the right arrow key to select "Finish" and press Enter to exit the configuration tool.
- Reboot your Raspberry Pi by typing:

```
sudo reboot
```

After the reboot, the camera interface will be enabled, and you can start developing your color recognition application. You can use programming languages like Python and libraries such as OpenCV to work with the camera and perform color recognition tasks.

Make sure you have the necessary software libraries installed by running the following commands:

```
sudo apt update
sudo apt install python3-opencv
```

V. connecting a Raspberry Pi 4 with the Raspberry Pi Camera V2 for the implementation of a banknote recognition system

To connect a Raspberry Pi 4 with the Raspberry Pi Camera V2 for the implementation of a banknote recognition system, you can follow these steps:

- Power off your Raspberry Pi 4 before connecting any hardware.
- Identify the camera connector on the Raspberry Pi 4. It is a small rectangular slot located near the HDMI port.
- Gently lift the plastic tab on the camera connector to open it.
- Take your Raspberry Pi Camera V2 and align the camera's flex cable with the camera connector on the Raspberry Pi
- Ensure that the metallic contacts on the flex cable face towards the HDMI port.
- Insert the flex cable into the camera connector, making sure it is inserted fully and evenly. Be careful not to apply excessive force.

- Once the flex cable is inserted correctly, gently push the plastic tab back down to secure the connection.
- Power on your Raspberry Pi 4.
- Open the terminal or SSH into your Raspberry Pi 4 and execute the following command to enable the camera interface:

  sudo raspi-config

  in the raspi-config menu, navigate to "Interfacing Options" and select "Camera." Then, choose "Enable" to activate the camera interface.
- Reboot your Raspberry Pi 4 for the changes to take effect.

After the reboot, you can start using the Raspberry Pi Camera V2 with your banknote recognition system. You can access the camera using Python and libraries like OpenCV or picamera to capture and process images or video.
We have the necessary software libraries and dependencies installed for your banknote recognition implementation.

VI. We also use to connect a Raspberry Pi 4 with the Raspberry Pi Camera V2 for the implementation of OCR and object detection.

3 Output

3.1 Headphone Output on Raspberry Pi 4

The Raspberry Pi 4 can be used with headphones as an audio output. The Raspberry Pi 4 has a 3.5mm audio jack that can be used to connect headphones or speakers.

To use headphones with your Raspberry Pi 4, follow these steps:

- Ensure that your Raspberry Pi 4 is powered off.
- Locate the 3.5mm audio jack on your Raspberry Pi 4. It is usually located next to the USB ports.

- Plug your headphones into the 3.5mm audio jack.
- Power on your Raspberry Pi 4.
  Once the Raspberry Pi 4 is powered on and booted up, you should be able to hear audio through your headphones.

By default, the audio output is set to the 3.5mm audio jack.

However, if you encounter any issues with audio playback, you can check the audio settings in the Raspberry Pi's configuration file. You can access the configuration file by opening the terminal and typing the following command:

```
sudo nano /boot/config.txt
```

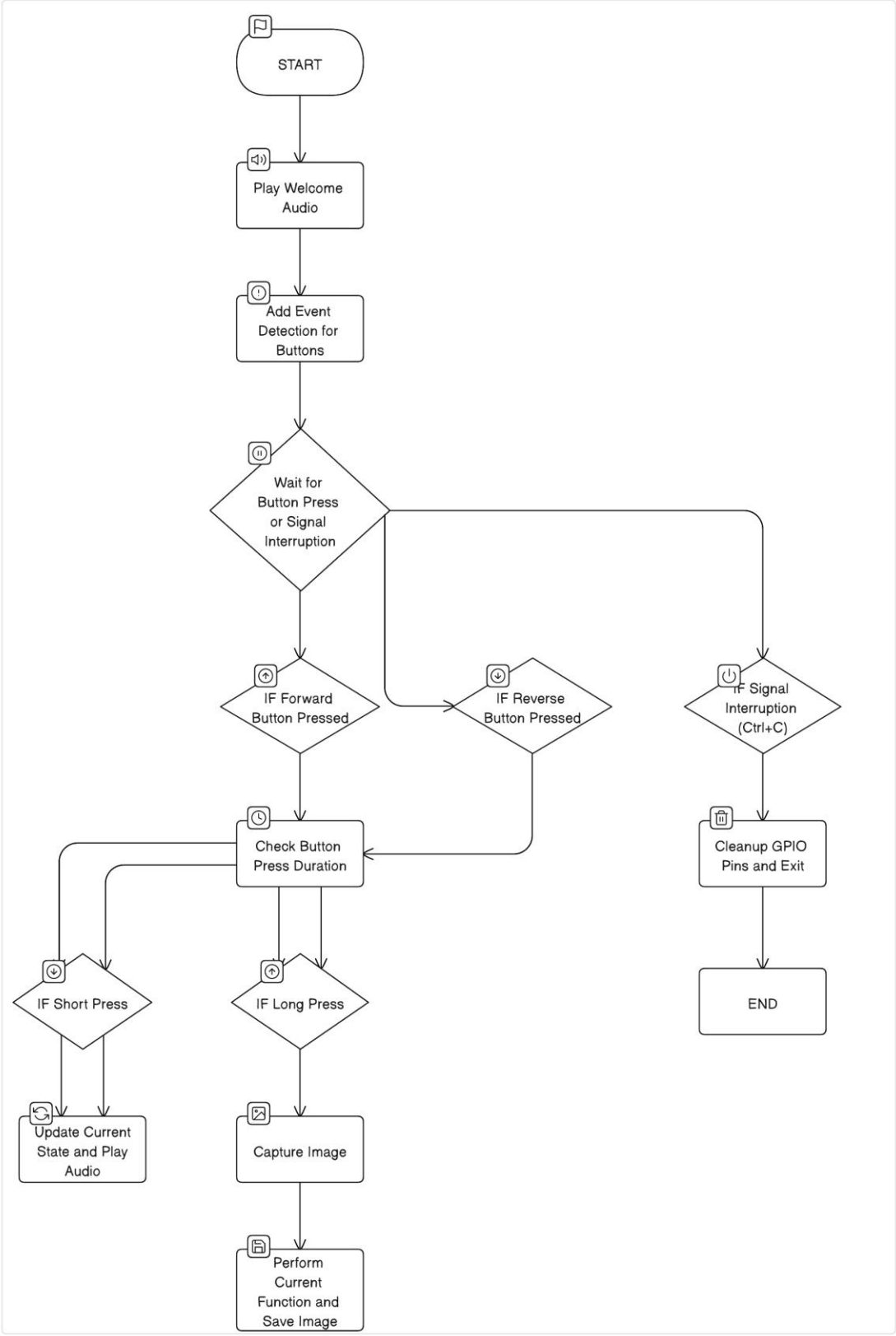
Look for the line that starts with dtparam=audio. Ensure that it is not commented out (no "#" symbol at the beginning of the line). If it is commented out, remove the "#" symbol and save the file.

After making any changes, you may need to reboot the Raspberry Pi for the changes to take effect. You can do this by typing the following command in the terminal:

```
sudo reboot
```

Once the Raspberry Pi reboots, audio should be directed to the headphones connected to the 3.5mm audio jack.

Flow chart



Implementation

This code appears to be a script written for a Raspberry Pi device that involves various functionalities such as camera operations, audio playback, GPIO button inputs, and different modes of operation.

1. The code starts by importing necessary libraries and modules such as `signal`, `sys`, `RPi.GPIO`, `os`, `time`, `picamera2`, `libcamera`, `numpy`, `cv2`, and `simpleaudio`. These modules provide functionalities for signal handling, system operations, GPIO control, camera operations, image processing, audio playback, and more.

2. Next, the code defines several global variables and dictionaries used throughout the script. These variables store information such as the current state, current language, number of states, GPIO pin numbers, audio paths, modes, and mappings for languages and modes.

3. The script defines several functions:

change_language(): This function is responsible for changing the current language. It updates the `current_language` variable and plays a welcome audio file in the new language.

perform_current_function(current_state, picam2, img, current_language) This function performs the appropriate action based on the current state. It takes the current state, the `picam2` object (for camera operations), the `img` image data, and the current language as inputs. Depending on the current state, it calls different functions related to face recognition, object detection, color recognition, banknote recognition, etc., and returns the result.

get_audio_path(audios_path, languages_map, current_language, modes_dir, modes_map, current_state)`: This function constructs the path to the audio file associated with the current state and language. It concatenates the necessary directories and filenames based on the provided inputs.

signal_handler(sig, frame)`: This function serves as the signal handler for the script. It performs cleanup operations on the GPIO pins and exits the script.

forward_button_callback(channel)`: This function is the callback for the forward button GPIO event. It handles both short and long presses of the button. In the case of a short press, it updates the current state and plays the corresponding audio file. In the case of a long press, it captures an image using the camera, performs the current function associated with the state, and saves the image.

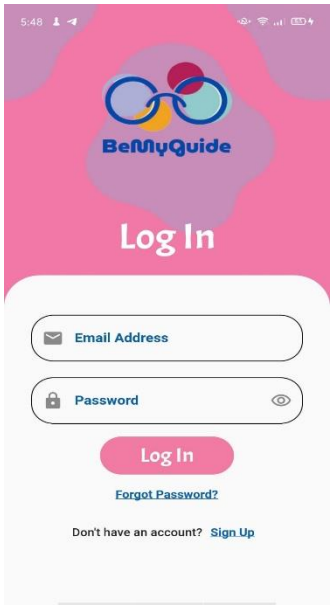
reverse_button_callback(channel)`: This function is the callback for the reverse button GPIO event. Similar to `forward_button_callback`, it handles short and long presses of the reverse button. In the case of a short press, it updates the current state and plays the corresponding audio file. In the case of a long press, it enables voice commands (not implemented in the provided code).
4. The script initializes the `picam2` object from the `Picamera2` class, sets up camera configurations, and starts the camera.
5. It sets up the GPIO mode and configures the forward and reverse buttons as inputs with pull-up resistors.
6. An audio file (welcome message) is played using the `simpleaudio` library.
7. The script adds event detection for the forward and reverse buttons Whenever a rising edge is detected on these GPIO pins, the respective callback functions (`forward_button_callback` and `reverse_button_callback`) are called.
8. The script sets up a signal handler for the SIGINT signal (Ctrl+C) to perform cleanup operations and exit gracefully.
9. Finally, the script enters a waiting state using `signal.pause()`, allowing the callbacks and signal handling to work in the background while the script remains active.
Overall, this script provides a framework for controlling a Raspberry Pi device with GPIO buttons, performing various functions based on button presses, capturing images, and interacting with audio files. The exact implementation of the functions such as face recognition, object detection, etc., is not provided in the code and would need to be implemented separately.

4.3 Mobile Application

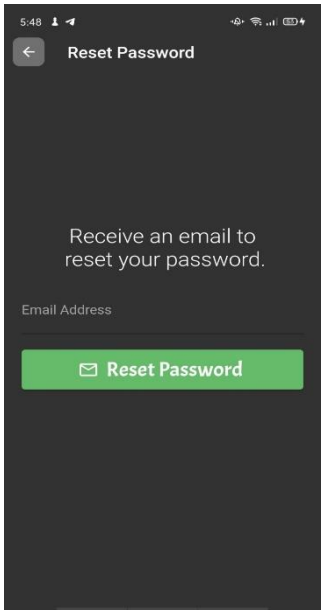
1- User interface

- Users can switch between languages Arabic and English.

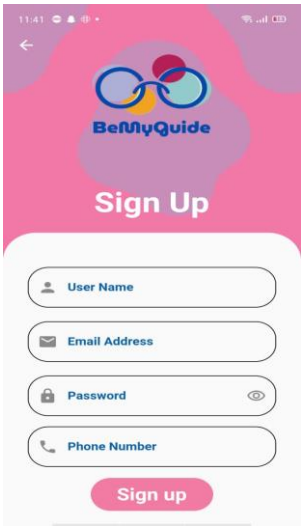
1) **Login Screen:** First Screen you will see, it allows you to put your email and password to enter the application or choose the sign up to create an account.
    - Validation is used to ensure that the user has entered valid data.
    - If the user entered wrong email or password a toast is shown with an error message.



2) **Reset Password Screen:** Screen that will appear when the user press Forgot Password, it used to send an email to the user in order to write a new password.



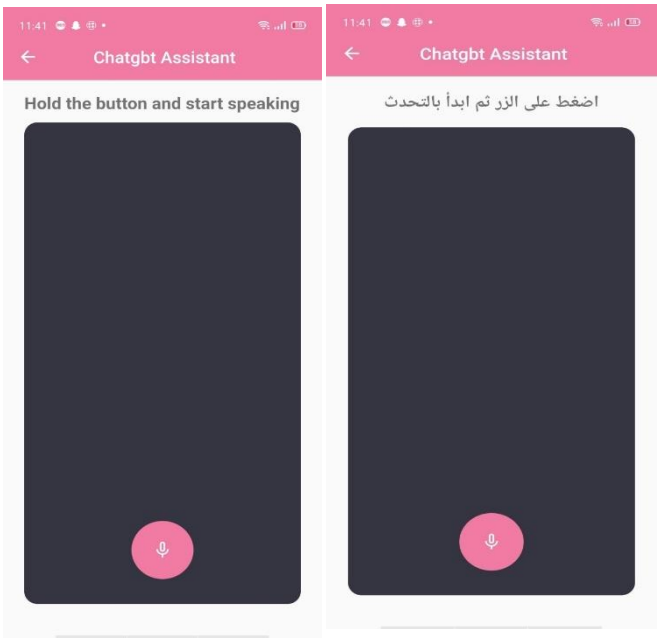
3) **Sign up Screen:** the screen which helps the user create an account.
    - Users enter their own data (name, email, password, phone number).
    - Validation is used to ensure that the user has entered valid data (email, the password should be at least 6 characters long).



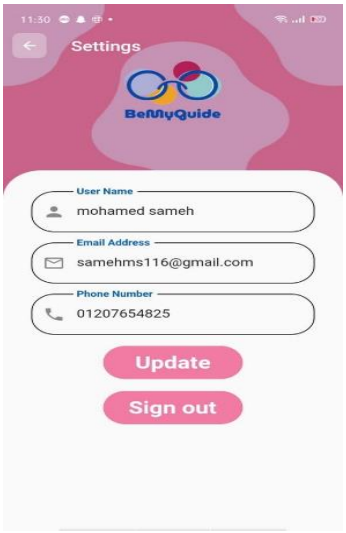
4) **Choose the mode screen:** The screen allows the user to choose whether they are here to volunteer or need some help.

5) **Assistant Mode:** If the user chose "Assistant", it provides several services:
    - Video call connection with Volunteers.
    - ChatGPT voice assistant in Arabic or English.
    - Sends a picture of the person who the user wants the glasses to recognize.
    - Google Maps to detect user location.

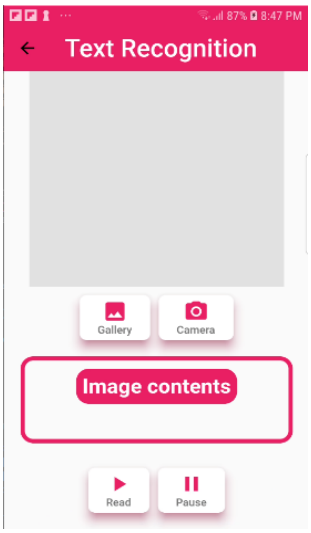
6) **ChatGPT Assistant Screen:** It provides ChatGPT services**.**
   - Users can communicate with the ChatGPT system using spoken language instead of typing.
   - The service supports both English and Arabic languages, allowing users to interact with the ChatGPT system in their preferred language.
   - This voice-based interaction provides a convenient and user-friendly way to utilize the ChatGPT services.



7) **Settings Screen:** It allows our users to update their info, or to sign out from the application.



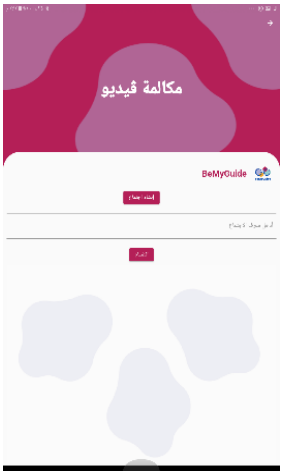
8) **Glasses Screen:** It allows our users to send photos from the app to the Glasses either from Gallery or Camera. This connection serves the purpose of utilizing the Glasses' capabilities, specifically in face recognition applications.



9) **Text Recognition Screen:** this screen is used for extracting text from photos. It also enables the app to read out the scanned text.

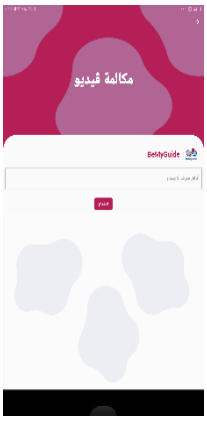
10) **Goole maps** inform user his location

11) **Video call:** user can create meeting or join meeting.



12) **Volunteer mode** is a specialized feature designed to support individuals who provide assistance to blind individuals. Contains video call and ChatGPT assistant.

13) Joining meeting as volunteer



2-Features

1. **ChatGPT:** First of all, what is ChatGPT?

- It is a state-of-the-art language model developed by OpenAI. It utilizes the power of deep learning to generate human-like responses to text prompts. It is designed to simulate natural language conversations and can be used in various applications such as chatbots, virtual assistants, customer support systems, and more.

- ChatGPT is trained on a diverse range of internet text data, allowing it to understand and generate responses across different topics and contexts. It leverages the knowledge it has learned during training to provide relevant and coherent answers.
- it's important to note that ChatGPT has some limitations. It can sometimes produce incorrect or nonsensical responses, and it may not always ask clarifying questions when faced with ambiguous queries. Care should be taken to validate and verify the generated responses, especially in critical or sensitive applications.

In our app, we have integrated the **OpenAI API** to incorporate ChatGPT functionality. However, instead of using text-based interactions, we have leveraged the TEXT-to-Speech package in Flutter to enable voice-based communication with ChatGPT. By utilizing the Text-to-Speech package, we allow users to input their queries or messages using voice commands. The voice input is converted to text format using Speech-to-Text package and then sent to the **ChatGPT API for processing.** The response generated by ChatGPT is then converted back into speech using the TEXT-to-Speech package, allowing users to hear the AI-generated responses.

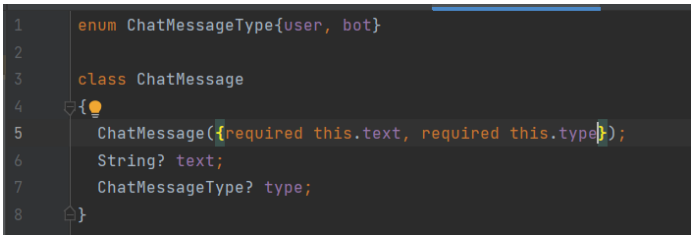


Figure 3

By combining the power of OpenAI's ChatGPT with the convenience of voice-based interaction, we aim to deliver a user-friendly and engaging experience in our application.

To differentiate between messages sent by the user and messages generated by

```
      var msg =  await ApiServices.sendMessage(text);
      msg = msg?.trim();
      // print(msg);
      setState(() {
        messages.add(ChatMessage(text: msg, type: ChatMessageType.bot));
      });

      Future.delayed(Duration(milliseconds: 400),() {
        TextToSpeech.speak(msg!);
      },
      );  // Future.delayed
    }else {
      ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text("Failed to process. Try again!")));
    }
```

ChatGPT.

Where the user's voice will be converted into text and then sent to ChatGPT.

```
child: GestureDetector(
  onTapDown: (details) async{
    if(!isListening)
    {
      bool av = await speechToText.initialize();
      if(av){
        setState(() {
          isListening = true;
        });
        speechToText.listen(
          localeId: selectedLocaleId,
          // pauseFor: Duration(
          //   minutes: 1,
          // ),
          // listenFor: Duration(
          //   minutes: 1,
          // ),
          onResult: (result) {
            setState(() {
              text = result.recognizedWords;
            });
          },
        );
      }
```

The Text-to-Speech class is responsible for converting the response from ChatGPT into voice.

```
import 'package:flutter_tts/flutter_tts.dart';

class TextToSpeech
{
  static FlutterTts tts = FlutterTts();
  static iniTts() async{
    // print(await tts.getLanguages);
    tts.setLanguage("en-US");
    tts.setPitch(1.0);
    // tts.setSpeechRate(0.2);
  }
  static speak(String text) async{

    tts.setStartHandler(() {
      print("TTS IS STARTED");
    });

    tts.setCompletionHandler(() {
      print("TTS IS COMPLETED");
    });

    tts.setErrorHandler((message) {
      print(message);
    });

    await tts.awaitSpeakCompletion(true);
    tts.speak(text);
  }
}
```

Figure 5

When we get the response, we wait for some delay to convert it.

Figure 6

2- Text Extraction and Text-to-Speech

This chapter focuses on the implementation of text extraction and text-to-speech functionalities in a Flutter application. We explore the process of extracting text from images using Google ML Kit, a powerful machine learning library. Additionally, we discuss the integration of text-to-speech capabilities using the flutter_tts package. Through practical examples and code snippets, this chapter aims to provide a comprehensive understanding of incorporating these features into a Flutter app.

1-Overview

1-Text Extraction:

Text extraction refers to the process of extracting text from various sources, such as images, documents, or web pages. In the context of a Flutter application, text extraction is typically used to extract text from images captured by the device's camera or selected from the gallery. This extraction process involves utilizing machine learning algorithms or optical character recognition (OCR) techniques to recognize and convert the text present in the images into a readable format.

2-Text-to-Speech:

Text-to-speech (TTS) is a technology that converts written text into spoken words. It enables devices and applications to audibly communicate textual information, providing an auditory representation of the text. In the context of a Flutter application, text-to-speech functionality allows users to have the extracted text or any other textual content within the app read out loud, providing an accessible and convenient way to consume information.

2. Importance and benefits of incorporating these features

1. Accessibility**:**

- Text extraction and text-to-speech features greatly enhance the accessibility of a Flutter application. These functionalities enable visually impaired users to access and comprehend textual content that may otherwise be inaccessible to them. By

extracting text from images and converting it to speech, users with visual impairments can effectively consume and interact with textual information.

2. Multimodal User Experience:

- By combining text extraction and text-to-speech functionalities, a Flutter application can provide a multimodal user experience. Users can both visually read the extracted text and listen to it being spoken aloud. This flexibility allows users to choose the mode of interaction that suits their preferences or specific situations, resulting in a more inclusive and user-friendly experience.

3. Increased Usability:

- Incorporating text extraction and text-to-speech features enhances the usability of an application. Users can quickly extract and comprehend information from images without the need for manual data entry or reading lengthy text blocks. Text-to-speech capabilities further improve usability by providing an alternative method of consuming textual content, reducing the cognitive load required for reading.

4. Language Localization:

- Text extraction and text-to-speech features facilitate language localization and global accessibility. By extracting and processing text from various sources, applications can support multiple languages and provide translations or language-specific functionalities. Text-to-speech capabilities enable users to listen to content in their preferred language, making the application more inclusive and globally accessible.

5. Enhanced Productivity and Efficiency:

- Text extraction features can streamline data entry and information retrieval processes within an application. Users can extract relevant information from documents, receipts, or images, saving time and effort required for manual data entry. Text-to-speech functionality allows users to consume information hands-free, increasing productivity and efficiency in various tasks, such as reading articles or studying.

Incorporating text extraction and text-to-speech features in a Flutter application brings significant benefits, including improved accessibility, enhanced usability, language localization, and increased productivity. These features cater to a broader range of users, provide alternative interaction modes, and streamline information consumption, ultimately enhancing the overall user experience of the application.

3. Objectives and scope of the chapter

The objectives of this chapter are to provide a comprehensive understanding of implementing text extraction and text-to-speech functionalities in a Flutter application. The chapter aims to cover the following key aspects:

1. Understanding Text Extraction:
   - Explain the concept of text extraction and its relevance in Flutter app development.
   - Introduce the Google ML Kit and its capabilities for text recognition.
   - Demonstrate the process of extracting text from images using Google ML Kit.
2. Implementing Text-to-Speech:
   - Introduce the concept of text-to-speech and its significance in enhancing app accessibility.
   - Discuss the flutter_tts package and its features for speech synthesis.
   - Guide the implementation of text-to-speech functionality in a Flutter application.
3. Integration and User Experience:
   - Explore the integration of text extraction and text-to-speech features within a Flutter app.
   - Discuss user interface considerations for displaying extracted text and enabling text-to-speech playback.
   - Address user interactions and provide guidelines for a seamless user experience.
4. Testing, Debugging, and Optimization:
   - Highlight the importance of testing text extraction accuracy and text-to-speech output quality.
   - Discuss debugging techniques and handling common issues related to these features.
   - Provide strategies for performance optimization and efficiency in text extraction and speech synthesis.
5. Best Practices and Future Trends:
   - Present best practices for implementing text extraction and text-to-speech functionalities.
   - Discuss considerations for accessibility compliance and localization.
   - Explore emerging trends and advancements in text processing and speech synthesis in Flutter.

The scope of this chapter is focused on the implementation of text extraction using Google ML Kit and text-to-speech functionality using the flutter_tts package in a Flutter application. It covers the technical aspects of integrating these features, addressing user interface considerations, testing, and debugging techniques, and providing best practices for efficient implementation. While the chapter provides a comprehensive understanding of these features, it does not delve into advanced topics such as natural language processing or advanced speech synthesis techniques.

3-Connetion between the app and the glasses (Mobile side)

This part focus on the integration of Raspberry Pi 4 with an Android application using FastAPI aims to establish a connection for sending photos from the app to the Raspberry Pi. This connection serves the purpose of utilizing the Raspberry Pi's capabilities, specifically in face recognition applications. By leveraging FastAPI's high-performance web framework and seamless communication, the Android application can securely transmit photos to the Raspberry Pi, enabling efficient processing and analysis for face recognition tasks. This integration provides a reliable and efficient solution for leveraging the power of the Raspberry Pi in real-time face recognition scenarios.

Conclusion:

In conclusion, the provided code demonstrates the integration of image selection and upload functionality within a Flutter app. The selected image(s) are uploaded to an API endpoint using Dio requests. This functionality enables users to interact with the app by selecting and uploading images, facilitating the exchange of data between the app and the API.

3-Video Call:

Firstly, generate unique token that will serve as a means of authenticating your identity within the VideoSDK ecosystem.

The token generation process is designed to be user-friendly and intuitive, allowing you to quickly and securely obtain the token required for using VideoSDK functionalities. The dashboard provides a seamless experience, guiding you through the necessary steps to generate your token with ease.

By leveraging the API section in the VideoSDK dashboard, users can effortlessly generate their tokens, enabling them to authenticate their identity and gain access

to the full range of VideoSDK features and capabilities. This streamlined process ensures a secure and efficient means of utilizing VideoSDK in your applications or services.

To generate a token, you can navigate to the VideoSDK dashboard and locate the API section. Within this section, you will find the necessary tools and options to generate your token.

This widget will contain JoinScreen and ILSScreen screens. ILSScreen will render the ILSSpeakerView or ILSViewerView based on the participants mode. ILSSpeakerView will have MeetingControls and ParticipantTile

Firstly, configure the project.

For Android

Update the /android/app/src/main/AndroidManifest.xml for the permissions we will be using to implement the audio and video features.

```
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.INTERNET"/>
```

- Also, you will need to set your build settings to Java 8 because the official WebRTC jar now uses static methods in EglBase interface. Just add this to your app-level /android/app/build.gradle.

```
android {
   //...
   compileOptions {
      sourceCompatibility JavaVersion.VERSION_1_8
      targetCompatibility JavaVersion.VERSION_1_8
   }
}
```

increase minSdkVersion of defaultConfig up to 23 and compileSdkVersion and targetSdkVersion up to 33.

For IOS

Add the following entries which allow app to access the camera and microphone to /ios/Runner/Info.plist file:

```
<key>NSCameraUsageDescription</key>
<string>$(PRODUCT_NAME) Camera Usage!</string>
<key>NSMicrophoneUsageDescription</key>
<string>$(PRODUCT_NAME) Microphone Usage!</string>
```

- api_call: build function generate unique meeting ID require token authentication.
- JoinScreen: this screen consists of create meeting button to create new meetings for the speaker. meeting ID textField contained the meeting Id you want to join. Join Meeting button to let other people join.
- Ils_screen: take the meetingId, and token. Create new room using createRoom methode.
- Participant_title: consist of RTCVideoView which show participant's video strem.
- Meeting_controls: this widget consists of meeting control buttons and the current HLS state of the meeting.
- Ils_speaker_view: this widget listens to hlsStateChanged, participantJoined and participantLeft events. It render the participants and the meeting controls like leave, toggle mic and webcam.
- Livestream_player: this widget use flutter_vlc_player package to play HLS stream on the viewer side.

4-Localizations

This app support two languages (Arabic and English) based on the language of users' software system. To use localization in flutter app:

1. Create folder "lang" contain 2 files and add in pubspec.yaml file:

```
dependencies:
  flutter_localizations:
    sdk: flutter
  localization: <last-version>

flutter:
  # json files directory this folder for translating screens
  assets:
    -assets/lang/
```

2. Add in main.dart file:

   a. localizationsDelegates to determine how many languages to add.
   b. supportedLocales: to add each language. Take language code and parameter code.
   c. localeResolutionCallback: allows to define a callback function that determines the app's locale based on the system locale and the list of supported locales. This function is responsible for resolving the desired locale and returning it to the app.

```
return MaterialApp(
  supportedLocales: [
    Locale('en', 'US'),
    Locale('es', 'ES'),
    Locale('pt', 'BR'),
  ],
    localizationsDelegates: [
       // delegate from flutter_localization
       GlobalMaterialLocalizations.delegate,
       GlobalWidgetsLocalizations.delegate,
       GlobalCupertinoLocalizations.delegate,
       // delegate from localization package.
       LocalJsonLocalization.delegate,
     ],
  home: HomePage(),
);
```

3. Create Applocale class to implements localization functionality in Flutter, including loading language files, providing translations, and supporting English and Arabic languages.
4. Open json files (en.json, ar.json) and starting to translate each screen to Arabic and English. Example:

```
en.json
{
“login” =”login”
}
```

```
ar.json
{
“login” =”تسجيل الدخول”
}
```

5. Update text in each screen using getLang() function which created in Applocale class. Example:

```
Text("${getLang(context,"login")}
```

5-Google Maps

This feature tell user his location using voice over implemented by using google maps API and google_maps_flutter package.

1. Generating API from google cloud server by enabling android apk apis and IOS.

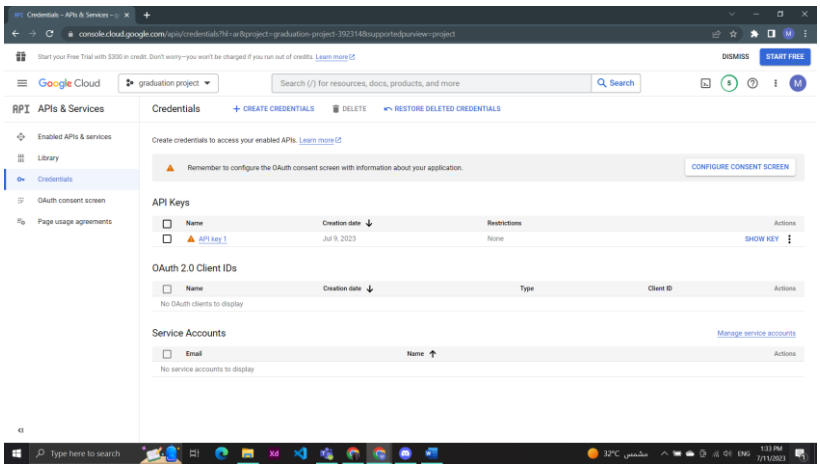


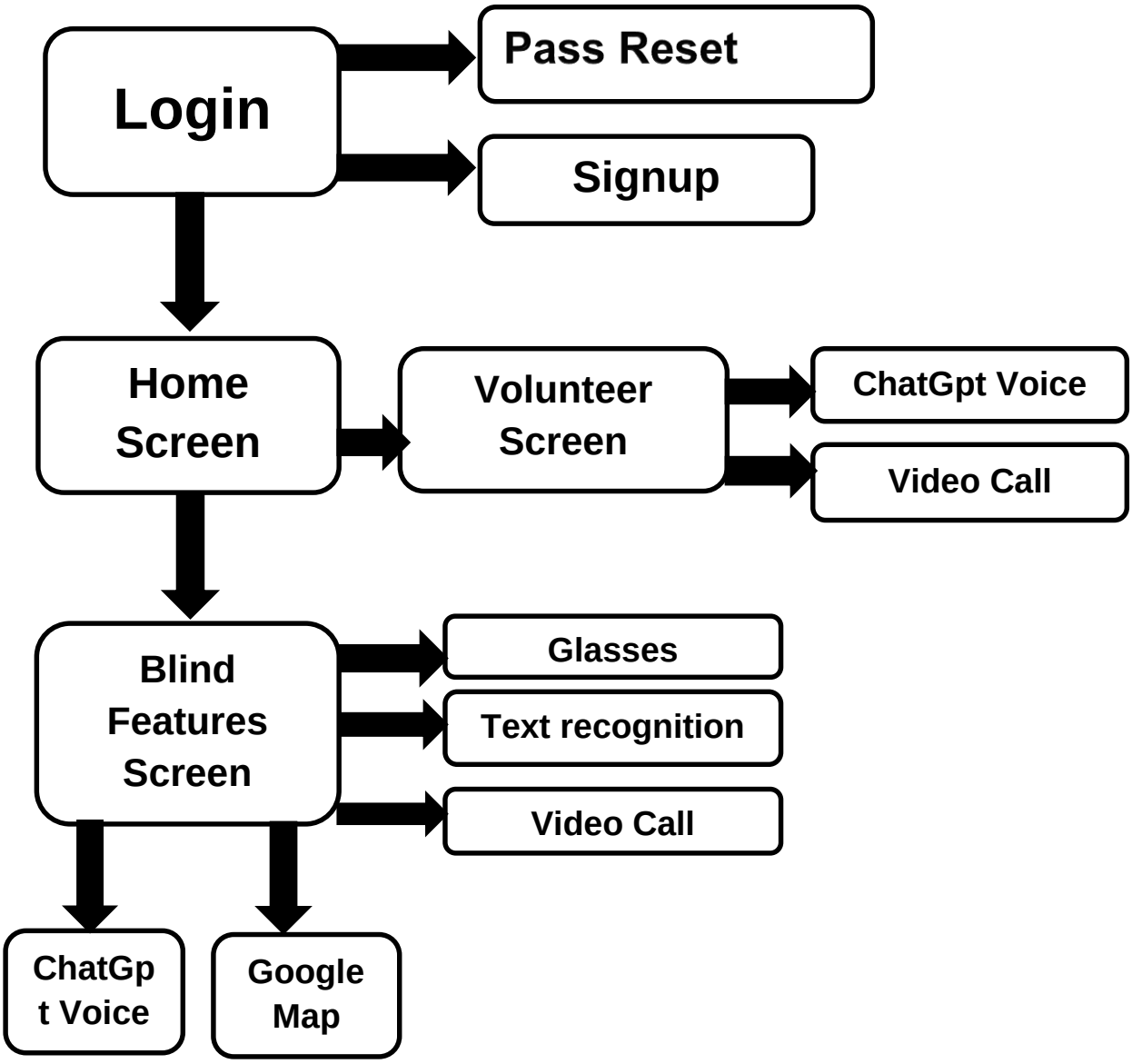
Figure 7

Specify your API key in the application manifest android/app/src/main/AndroidManifest.xml:

```
<manifest ...
  <application ...
    <meta-data android:name="com.google.android.geo.API_KEY"
               android:value="APIKey"/>
```

2. Set up google maps package in pubspec.yaml file.
3. Create a google maps widget to determine the location of users and their live location, then use the geocoding package to convert coordinates into address.

3-Application Structure:


Login
Pass Reset
Signup
Home Screen
Volunteer Screen
ChatGpt Voice
Video Call
Blind Features Screen
Glasses
Text recognition
Video Call
ChatGp t Voice
Google Map

4.4Connecting Raspberry Pi 4 with Android Application using FastAPI

The integration of Raspberry Pi 4 with an Android application using FastAPI aims to establish a connection for sending photos from the app to the Raspberry Pi. This connection serves the purpose of utilizing the Raspberry Pi's capabilities, specifically in face recognition applications. By leveraging FastAPI's high-performance web framework and seamless communication, the Android application can securely transmit photos to the Raspberry Pi, enabling efficient processing and analysis for face recognition tasks. This integration provides a reliable and efficient solution for leveraging the power of the Raspberry Pi in real-time face recognition scenarios.

I. FastAPI

i. Introduction and Benefits

FastAPI is a modern, high-performance web framework for building APIs with Python. It offers several advantages that make it an excellent choice for connecting a Raspberry Pi 4 with an Android application.

ii. Performance and Efficiency

FastAPI leverages asynchronous programming and utilizes the capabilities of modern Python frameworks such as Starlette and Pydantic. This results in highly performant APIs that can handle a large number of concurrent connections, making it suitable for real-time communication between the Raspberry Pi and the Android device.

iii. Automatic Documentation and Validation

FastAPI generates interactive API documentation automatically, based on the function and model definitions. It includes detailed information about request and response types, making it easier to understand and use the API. Additionally, FastAPI performs data validation and serialization, ensuring the integrity and consistency of data exchanged between the Raspberry Pi and the Android application.

iv. Type Hints and Editor Support

FastAPI utilizes Python type hints to improve code readability and provides enhanced editor support, including autocompletion and error checking. This feature aids in developing the connection between the Raspberry Pi and the Android application efficiently, minimizing errors and enhancing developer productivity.

v. Security and Authentication

FastAPI offers robust security features, including built-in support for OAuth2, JWT (JSON Web Tokens), and HTTPS. These features enable secure communication between the Raspberry Pi and the Android application, ensuring data privacy and integrity.

II. Advantages of FastAPI over Bluetooth and Cloud-based Solutions

i. Faster data transfer and improved performance: FastAPI utilizes the local network connection, which typically offers higher bandwidth and lower latency compared to Bluetooth. This results in faster data transfer, enabling efficient communication between the Android app and Raspberry Pi.

ii. Seamless integration within the local network: FastAPI allows direct communication between the Android app and Raspberry Pi within the same local network. This eliminates the need for cloud servers and reduces dependencies on external services, simplifying the setup and reducing potential points of failure.

iii. Local network communication without relying on internet connectivity: With FastAPI, the Android app and Raspberry Pi can communicate directly over the local network without requiring an internet connection. This provides greater flexibility and stability, especially in environments where internet connectivity may be limited or unreliable.

iv. Full control and customization of interactions: FastAPI provides a flexible framework for designing and implementing custom interactions between

the Android app and Raspberry Pi. Developers have full control over the communication protocol, data formats, and message handling, allowing for tailored and optimized solutions.

v. Enhanced privacy and data security: By utilizing local network communication, FastAPI reduces the exposure of sensitive data to external servers and cloud platforms. This enhances privacy and data security, as data transmission occurs within the confines of the local network, mitigating potential risks associated with cloud-based solutions.

vi. Cost-effectiveness: FastAPI leverages existing infrastructure within the local network, eliminating the need for additional hardware or subscription costs associated with cloud-based services. This makes it a cost-effective solution, particularly for projects with budget constraints.

Note: The usage of Bluetooth causes problems with a Wi-Fi connection, as both Bluetooth and Wi-Fi operate in the 2.4 GHz frequency range, which means they share the same frequency band. This leads to interference between the two technologies, resulting in decreased performance or disrupted connectivity.

To mitigate potential interference issues, it is advisable to keep Bluetooth and Wi-Fi devices as far apart as possible, reduce the number of active Bluetooth devices, or use devices that support advanced coexistence mechanisms. Additionally, using less congested Wi-Fi channels or operating in the 5 GHz frequency range, if supported, can help minimize interference.

III. Establishing the Connection

The steps for establishing the connection between the Raspberry Pi 4 and the Android application using FastAPI for photo transfer in face recognition:

i. Setting up the Raspberry Pi: Ensure that the Raspberry Pi 4 is properly configured and connected to the local network. This typically involves connecting it to a monitor, keyboard, and mouse, and configuring the network settings.

ii. Installing FastAPI: Install FastAPI on the Raspberry Pi by following the official documentation. This involves using package managers like pip to

```
pip3 install fastapi
pip3 install "uvicorn[standard]"
```

install the necessary dependencies.

iii. Developing the API: Create a FastAPI application on the Raspberry Pi that includes the necessary endpoints for receiving photos from the Android application.

The endpoint handles a POST request to the /upload URL. Here's a description of what it does:

- When a POST request is made to /upload, it expects a file to be attached with the request.
- The UploadFile parameter file represents the uploaded file. FastAPI automatically handles the file upload and provides it as an instance of the UploadFile class.
- Then creates a new file in the "media" directory with the same name as the uploaded file using open(). It opens the file in binary write mode ('wb').
- The await file.read() statement reads the contents of the uploaded file asynchronously.
- The file contents are then written to the newly created file using f.write(data).
- Finally, the endpoint returns a JSON response with {'done': True} to indicate that the file upload is completed successfully.

By using a systemd service configuration file for running a FastAPI application on a Raspberry Pi, it specifies the user, working directory, and command to start the application using uvicorn. The service is set to automatically restart if it stops unexpectedly, and it is configured to be enabled and started during system initialization in multi-user mode. By using this configuration, you can manage your FastAPI application as a service on your Raspberry Pi, allowing for easier control and monitoring of the application's execution.

IV. Configuring the Android Application: In the Android application, configure the network settings to establish a connection with the Raspberry Pi. This may involve specifying the IP address or hostname of the Raspberry Pi within the local network.

V. Implementing Photo Transfer: Develop the functionality in the Android application to capture or select photos and send them to the Raspberry Pi using FastAPI's API endpoints. Ensure proper serialization and formatting of the photo data for seamless transmission.

Appendix

1Artificial Intelligence & Computer Vision codes:

- **Banknote:**

```
import onnxruntime as ort

import numpy as np

import cv2



class BanknotesRecognizer(object):

    def __init__(self,

                 model_path,

                 img_shape,

                 confidence_score=0.8,

                 tta=False

                 ):


        self.model = ort.InferenceSession(model_path)

        self.img_shape = img_shape

        self.confidence_score = confidence_score

        self.tta = tta
```

```
def __call__(self,img):
    img = self.prepare_img(img)
    return self.forward(img)


def softmax(self,x):
    return np.exp(x)/np.sum(np.exp(x))


def normalize_image(self,img,mean,std):
    img = img/255.
    img = (img-mean)/std
    return img.astype(np.float32)


def prepare_img(self,img):
    img = cv2.resize(img,self.img_shape)
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    img = self.normalize_image(img,
```

```
                [0.485, 0.456, 0.406],
                [0.229, 0.224, 0.225])
        img = np.transpose(img,(2,0,1))
        img = np.expand_dims(img,axis=0)
        return img


    def forward(self,img):
        out = self.model.run(None,{self.model.get_inputs()[0].name:img})
        return self.softmax(out[0])
```

- **Face detection code:**

```
    import os
    import cv2
    import numpy as np
    # from albumentations import rotate,normalize



    class FaceDetector(object):
```

```
def __init__(self
        ,model_path=r'./face_lib/models/detector.onnx'
        ,conf_th=0.6
        ,nms_th=0.3
        ,input_size=(320,320)
        ):

    self.input_size = input_size
    self.detector = cv2.FaceDetectorYN.create(model_path, "",(0,0))
    self.detector.setScoreThreshold(conf_th)
    self.detector.setNMSThreshold(nms_th)
    self.detector.setInputSize(self.input_size)



def __call__(self,img,face_size=(112,112)):
    return self.extract_faces(img,face_size)
```

```
def _get_scale_factor(self,img):
    """
    Gets the scaling factor between the dimentions
    of the actual image and the resized one.
    """
    return (np.zeros((14)).reshape(-1,2)
    + np.array([img.shape[1],img.shape[0]])
    /np.array((self.input_size[1],self.input_size[0]))).reshape(14)


def _scale_keypoints(self,img,keypoints):
    '''
     Scales the faces keypoints' locations obtained from the resized image
     to match their actual locations in the full scale image.
     '''
    return keypoints[:,:-1]*self._get_scale_factor(img)


def _detect_faces(self,img):
    '''
```

```
        Performs detection.
        returns arranged keypoints.
        '''
        img_copy = cv2.resize(img,self.input_size)
        out = self.detector.detect(img_copy)
        if(out[1] is not None):
            detections = out[1]
            detections[detections < 0] = 0
            keypoints = self._scale_keypoints(img,detections)
            return
np.rint(keypoints[:,:4]).astype(int),np.rint(keypoints[:,4:]).reshape(-
1,5,2).astype(int)
        return None



    def _rotate(self,img,p1,p2,direction):


        dx = p2[0] - p1[0]
        dy = p2[1] - p1[1]
        angle = np.degrees(np.arctan2(dy, dx)) - 180
```

```
    rotation_matrix = np.array([[],
                                [],],dtype=np.float32)



def _vertically_align(self,face,facial_points):
    '''
    vertically align the face based on the predicted position of the eyes.
    '''

    (x1,y1) = facial_points[0]
    (x2,y2) = facial_points[1]

    if(y1==y2): #no rotation required
        return face

    if(y1>y2):
        self._rotate_clockwise()
```

```
    def _extract_face(self,img,bbox,facial_keypoints,face_size,align=False):

        face = img[bbox[1]:bbox[1]+bbox[3],bbox[0]:bbox[0]+bbox[2]]

        if(align):
            face = self._vertically_align(face = face,facial_keypoints =
facial_keypoints)

        face = cv2.resize(face,face_size)
        bbox = bbox
        facial_keypoints = {'left eye' : facial_keypoints[0],
                            'right eye' : facial_keypoints[1],
                            'nose' : facial_keypoints[2],
                            'left mouth' : facial_keypoints[3],
                            'right mouth' : facial_keypoints[4]}
        return {'face':face,
                'bbox':bbox,
                'facial_keypoints':facial_keypoints}
```

```
    def extract_faces(self,img,face_size=(112,112)):
        out = self._detect_faces(img)
        if(out is not None):
            bboxes,facial_keypoints = out
            img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
            faces_data = {}
            for i in range(len(bboxes)):
                face_data =
self._extract_face(img,bboxes[i],facial_keypoints[i],face_size,False)
                faces_data[i] = face_data
            return faces_data
        return None
```

- Face recognition:

```
import os
import cv2
import tflite_runtime.interpreter as tflite
import numpy as np
from numpy.linalg import norm
import time
```

```
class FaceRecognizer(object):

    def __init__(self,
                 model_path=r'./face_lib/models/recognizer.tflite',
                 distance_metric='l2',
                 threshold=0.8,
                 dtype = np.float32):

        self.model_path = model_path
        self.interpreter = self._create_interpreter()
        if(distance_metric == 'cosine'):
            self.distance_metric = 'cosine'
        else:
            self.distance_metric = 'l2'

        self.threshold = threshold
        self.dtype = dtype

        #details of input and output tensors
        self.tensor_input_details = self.interpreter.get_input_details()
```

```
        self.tensor_output_details = self.interpreter.get_output_details()

        self.input_shape = self.tensor_input_details[0]['shape']



    def _create_interpreter(self):


        interpreter = tflite.Interpreter(model_path=self.model_path)

        interpreter.allocate_tensors()

        return interpreter



    def get_embeddings(self,face_img):

        input = ((face_img.reshape(self.input_shape)-
127.5)/127.5).astype(self.dtype)

        self.interpreter.set_tensor(self.tensor_input_details[0]['index'],input)

        self.interpreter.invoke()

        embeddings =
self.interpreter.get_tensor(self.tensor_output_details[0]['index'])

        if(self.distance_metric=='cosine'):

            return embeddings/norm(embeddings).reshape(-1) #normalize the
embeddings for computational effeciency.
```

```
        else:
            return embeddings.reshape(-1)


    def _cosine_distance(self,face_embeddings,embeddings_bank):
        return (1-np.dot(embeddings_bank,face_embeddings))


    def _euclidean_distance(self,face_embeddings,embeddings_bank):
        diff = np.subtract(embeddings_bank,face_embeddings)
        dist = np.sqrt(np.sum(diff*diff,axis=1))
        return dist


    def compute_distance(self,embeddings1,embeddings2):
        if(self.distance_metric=='cosine'):
            return
self._cosine_distance(face_embeddings=embeddings1,embeddings_bank=
embeddings2)
```

```
        return
self._euclidean_distance(face_embeddings=embeddings1,embeddings_ban
k=embeddings2)



    def compare(self,img1,img2,threshold=None):

        emb1 = self.get_embeddings(img1)

        emb2 = self.get_embeddings(img2)

        dist = self.compute_distance(emb1,emb2)


        if(threshold is None and dist<=self.threshold):

            return 1

        else:

            return 0



    def identify_face(self,face_img,faces_bank):

        if(face_img is None or faces_bank is None):

            return None


        faces_embeddings = faces_bank[0]
```

```
        indices = faces_bank[1]

        embeddings = self.get_embeddings(face_img)

        distances =
    self.compute_distance(embeddings1=embeddings,embeddings2=faces_em
    beddings).reshape(-1)

        min_index = int(distances.argmin())

        if(distances[min_index]<= self.threshold):

            return int(indices[min_index])

        else:

            return -1



    def identify_faces(self,faces,faces_bank):

        if(faces is None or faces_bank is None):

            return None


        indices = []

        for key in faces:

            indices.append(self.identify_face(faces[key]['face'],faces_bank))


                return indices
```

- **Face pipeline:**

```
import os
import cv2
import numpy as np
from .detect import FaceDetector
from .recognize import FaceRecognizer
from .faces_data import FacesData



class FaceIdentificationPipeline(object):
    def __init__(self,detector,recognizer,data):

        if(isinstance(detector,FaceDetector)):
            self.detector = detector
        else:
            raise Exception("detector must be an instance of FaceDetector")

        if(isinstance(recognizer,FaceRecognizer)):
            self.recognizer = recognizer
```

```
        else:
            raise Exception("recognizer must be an instance of
FaceRecognizer")

        if(isinstance(data,FacesData)):
            self.data = data
        else:
            raise Exception("data must be an instance of FacesData")


    def get_faces(self,frame):
        return self.detector.extract_faces(frame)


    def get_faces_embeddings(self,frame):
        faces = self.get_faces(frame)
        if(faces is None):
            return None
        embeddings = []
```

```
        for key in faces:

embeddings.append(self.recognizer.get_embeddings(faces[key]['face']))

        if (len(embeddings)==1):
            return embeddings[0]

        return embeddings


    def identify_faces(self,frame):
        faces_bank = self.data.get_faces_bank()
        if(faces_bank is None):
            return None

        faces = self.get_faces(frame)

        if(faces is None):
            return None
```

```
        return self.recognizer.identify_faces(faces,faces_bank) #-1 means
    unkown



    def identify_face_audiofile(self,face_img,faces_bank,audios_dir):
        idx = self.identify_faces(face_img)
        if(idx==-1):
            return None
        elif(idx is None):
            return None
        else:
            file_name = os.path.join(audios_dir,str(idx[0])+'.wav')
            return idx[0],file_name



    def identify_faces_audiofiles(self,faces):
        faces_bank = self.data.get_faces_bank()
        audios_dir = self.data.audios_dir
        if(faces_bank is None):
            return None
```

```
        indices = []
        files = []
        for key in faces:

            idx,file_name =
self.identify_face_audiofile(faces[key]['face'],faces_bank,audios_dir)
            if(idx==-1):
                continue
            indices.append(idx)
            files.append(file_name)

        return indices,files


    def register_face(self,img,name,lang):
        if(type(img) == np.ndarray):
            embeddings = self.get_faces_embeddings(img)
        f(type(img) == str):
            img = cv2.imread(img)
            embeddings = self.get_faces_embeddings(img)
```

```
        return self.data.register_face(embeddings,name,lang)
```

- Faces_data:

```
import os

# import glob

# import wave

# import simpleaudio as sa

from pydub import AudioSegment

# import cv2

import numpy as np

from numpy import genfromtxt

from gtts import gTTS

import csv

import json



class FacesData(object):

    def __init__(self,faces_bank_file="faces_bank.csv",

                 registered_file = "registered.json",

                 audios_dir = 'audios'):

        self.audios_dir = audios_dir
```

```
        self.faces_bank_file = faces_bank_file
        self.registered_file = registered_file
        self.registered = self.get_registered()
        self.current_idx = self.get_current_idx()


    def get_registered(self):

        if(not os.path.exists(self.registered_file)):
            with open(self.registered_file,"a") as f:
                json.dump({},f)

        with open(self.registered_file,"r") as f:
            return json.load(f)


    def get_current_idx(self):
        data = self.get_faces_bank()
        if(data is None):
            current_idx = 0
```

```
        else:
            current_idx = data[1].max()

        return current_idx


    def write_data(self,data):

        with open(self.faces_bank_file,'a') as file:
            writer  = csv.writer(file)
            writer.writerow(data)


    def get_faces_bank(self):
        if(not os.path.exists(self.faces_bank_file)):
            with open(self.faces_bank_file,'a')as f:
                pass
            return None

        data = genfromtxt(self.faces_bank_file, delimiter=',')
```

```
        if(not data.any()):
            return None

        if(len(data.shape)==2):
            faces_embeddings = data[:,:-1]
            indices = data[:,-1].astype(int)

        else:
            faces_embeddings = data[:-1].reshape(1,-1)
            indices = data[-1].astype(int).reshape(1,-1)

        return faces_embeddings,indices



    def _is_registered(self,name):
        if(name in self.registered):
            return True
        return False
```

```
    def register_face(self,embeddings,name,lang='en'):

        if(not os.path.exists(self.audios_dir)):
            os.mkdir(self.audios_dir)
        data = np.zeros(embeddings.shape[0]+1)
        data[:-1] = embeddings
        if(self._is_registered(name)):
            data[-1] = self.registered[name]
        else:
            self.current_idx += 1
            data[-1] = self.current_idx
            self.registered[name] = int(self.current_idx)
            with open(self.registered_file,"w") as f:
                json.dump(self.registered,f)
                speech = gTTS(name,lang=lang)
                file_name_wav =
os.path.join(self.audios_dir,str(self.registered[name])+'.wav')
                file_name_mp3 =
os.path.join(self.audios_dir,str(self.registered[name])+'.mp3')
```

```
            speech.save(file_name_mp3)
            sound = AudioSegment.from_mp3(file_name_mp3)
            sound.export(file_name_wav, format="wav")

        self.write_data(data=data)

        return 1
```

- Color recognition:

```
import cv2
    import numpy as np
    from numpy import genfromtxt

    class ColorRecognizer(object):
      def __init__(self,colors_file):

        self.colors_data = genfromtxt(colors_file, delimiter=',')

        self.colors_map = {0:'sky blue',
                           1:'dark blue',
```

```
                2:'blue',
                3:'turqoise',
                4:'maroon',
                5:'red',
                6:'yellow',
                7:'orange',
                8:'olive',
                9:'green',
                10:'gray',
                11:'silver',
                12:'pink',
                13:'purple',
                14:'brown',
                15:'beige',
                16:'white',
                17:'black'}


    def __call__(self,img):
        img = self.preprocess(img)
```

```
        img_shape = img.shape[:-1]

        batch = img[img_shape[0]//2-
25:img_shape[0]//2+25,img_shape[1]//2-25:img_shape[1]//2+25]

        batch = cv2.fastNlMeansDenoisingColored(batch,None,10,10,7,21)

        r = np.mean(batch[:,:,0])

        g = np.mean(batch[:,:,1])

        b = np.mean(batch[:,:,2])

        test_data = np.array([r.item(),g.item(),b.item()])



        return self.get_label(test_data=test_data)



    def preprocess(self,img):

        img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)


        return img



    def calc_distances(self,test_data):

        diff = self.colors_data[:,:-1]-test_data
```

```
        dist = np.sum(diff*diff,axis=1)
        return dist


    def get_label(self,test_data):
        labels = self.colors_data[:,-1]
        distances = self.calc_distances(test_data)
        return self.colors_map[labels[np.argmin(distances,axis=0)]]
```

- **Object detection:**

```
import onnxruntime as ort
import numpy as np
import cv2


class ObjectDetector(object):


    def __init__(self,
                 model_path,
                 classes_file,
```

```
            img_shape,
            conf_threshold,
            iou_threshold):

        self.model = ort.InferenceSession(model_path)
        self.classes_file = classes_file
        self.classes = self.get_classes()
        self.img_shape = img_shape
        self.conf_threshld = conf_threshold
        self.iou_threshold = iou_threshold


    def get_classes(self):
        if(self.classes_file is None):
            return None

        with open(self.classes_file,'r') as f:
            classes = [line.strip() for line in f.readlines()]
        return classes
```

```
def prepare_img(self,img):

    img = cv2.resize(img,self.img_shape)
    img = img/255.
    img = np.transpose(img,(2,0,1))
    img = np.expand_dims(img,axis=0)
    return img.astype(np.float32)


def preprosess(self,img):

    original_image = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    [height, width, _] = original_image.shape
    length = max((height, width))
    image = np.zeros((length, length, 3), np.uint8)
    image[0:height, 0:width] = original_image
    scale = length / self.img_shape[0]
    return self.prepare_img(image)
```

```
    def postprocess(self,outputs):

        rows = outputs.shape[1]

        boxes = []
        scores = []
        class_ids = []

        for i in range(rows):
            classes_scores = outputs[0][i][4:]
            (minScore, maxScore, minClassLoc, (x, maxClassIndex)) =
cv2.minMaxLoc(classes_scores)
            if maxScore >= 0.25:
                box = [
                    outputs[0][i][0] - (0.5 * outputs[0][i][2]), outputs[0][i][1] -
(0.5 * outputs[0][i][3]),
                    outputs[0][i][2], outputs[0][i][3]]
                boxes.append(box)
                scores.append(maxScore)
                class_ids.append(maxClassIndex)
```

```
        result_boxes = cv2.dnn.NMSBoxes(boxes, scores, self.conf_threshld,
self.iou_threshold)


        detections = []
        for i in range(len(result_boxes)):
            index = result_boxes[i]
            box = boxes[index]
            detection = {
                'class_id': class_ids[index],
                'class_name': self.classes[class_ids[index]],
                'confidence': scores[index],
                'box': box,
                }
            detections.append(detection)
        return detections


    def __call__(self,img):
        inp = self.preprosess(img)
        out = self.forward(inp)
```

```
        detections = self.postprocess(out)

        return detections




    def forward(self,img):


        out = self.model.run(None,{self.model.get_inputs()[0].name:img})

        return np.transpose(out[0],(0,2,1))
```

- **OCR:**

```
# import the necessary packages
import os
import glob
import io
import json
import time
import requests
import simpleaudio as sa
from pydub import AudioSegment
from gtts import gTTS
import langdetect
from azure.cognitiveservices.vision.computervision import
ComputerVisionClient
from azure.cognitiveservices.vision.computervision.models import
OperationStatusCodes
from msrest.authentication import CognitiveServicesCredentials
```

```
def init():
    credential = json.load(open('credential.json'))
    API_KEY = credential['API_KEY']
    ENDPOINT = credential['ENDPOINT']

    #analyzing the image
    cv_client =
ComputerVisionClient(ENDPOINT,CognitiveServicesCredentials(API_K
EY))
    return cv_client

def ocr(image_path,mode="fast"):
    client = init()
    response = client.read_in_stream(open(image_path, 'rb'),
                                     raw = True)
    OperationLocation = response.headers['Operation-Location']
    operationID = OperationLocation.split('/')[-1]

    while True:
        # get the result
        results = client.get_read_result(operationID)
        # check if the status is not "not started" or "running", if so,
        # stop the polling operation
        if results.status.lower() not in ["notstarted", "running"]:
            break
        # sleep for a bit before we make another request to the API
        time.sleep(1)
    # check to see if the request succeeded
    if results.status == OperationStatusCodes.succeeded:
        # print("[INFO] Microsoft Cognitive Services API request
succeeded...")
        for result in results.analyze_result.read_results:
        # loop over the lines
            texts = []
            for line in result.lines:
                # extract the OCR'd line from Microsoft's API and unpack the
                # bounding box coordinates
                text = line.text
                texts.append(text)
            if(mode=="document"):
```

```
                with open("text.txt","a",encoding="utf-8") as f:
                    f.write(text)
            else:
                return texts
            # if the request failed, show an error message and exit

        else:
            print("[INFO] Microsoft Cognitive Services API request failed")
            print("[INFO] Attempting to gracefully exit")


    def text_to_speech(texts):
        i = 0
        for text in texts:
            mp3_file_name = str(i)+".mp3"
            wav_file_name = str(i)+".wav"

            if(text.isnumeric()):
                speech = gTTS(text, lang='en', slow=False, tld='com')

            elif(langdetect.detect(text)=="ar"):
                speech = gTTS(text, lang='ar', slow=False, tld='com')
            else:
                speech = gTTS(text, lang='en', slow=False, tld='com')


            try:
                speech.save(mp3_file_name)
            except:
                continue
            sound = AudioSegment.from_mp3(mp3_file_name)
            sound.export(wav_file_name, format="wav")
            os.remove(mp3_file_name)
            i+=1

    def play_audios(path):
        audios = glob.glob("*.wav")
        i = 0
        for audio in audios:
            wave_obj = sa.WaveObject.from_wave_file(audio)
```

```
        play_obj = wave_obj.play()
        play_obj.wait_done()
        i+=1
```

2 - python code

```
import signal
import sys
import RPi.GPIO as GPIO
import os
import time

from picamera2 import Picamera2
from libcamera import Transform
import numpy as np
import cv2

import simpleaudio as sa
from banknotes_rec import recognize_banknote,classifier
from color_rec import recognize_color,color_recognizer
from object_det import detect_objects,object_detector
from face_rec import recognize_faces,pipeline


current_state = 0
current_language = 0
num_states = 7
FORWARD_BUTTON_GPIO = 2
REVERSE_BUTTON_GPIO = 3
s = 0
press_threshold = 0.4
threshold = 1.5


audios_path = '/home/pi/python/project/data/audios'
```

```
modes_dir = 'modes'

languages_map = {
    0:'ar',
    1:'en'
    }

modes_map = {
    0:'welcome',
    1:'face_recognition',
    2:'object_recognition',
    3:'color_recognition',
    4:'banknote_recognition',
    5:'read_text',
    6:'read_document',
    7:'change_language',
}

def change_language():
    global current_language
    if(current_language==0):
        current_language = 1
    elif(current_language==1):
        current_language = 0
    wave_obj =
sa.WaveObject.from_wave_file(f'{audios_path}'+'/'+f'{languages_map[cur
rent_language]}'+'/modes/welcome.wav')
    play_obj = wave_obj.play()
    play_obj.wait_done()
    return f"language changed to {languages_map[current_language]}"


def
perform_current_function(current_state,picam2,img,current_language):
    if(current_state==1):
        return recognize_faces(pipeline,img)
    elif(current_state==2):
        return detect_objects(object_detector,img)
    elif(current_state==3):
        return recognize_color(color_recognizer,img)
```

```
    elif(current_state==4):
        return recognize_banknote(classifier,picam2)
    elif(current_state==5):
        return None
    elif(current_state==6):
        return None
    elif(current_state==7):
        return change_language()


def
get_audio_path(audios_path,languages_map,current_language,modes_dir,
modes_map,current_state):
    audio_path = os.path.join(audios_path,
                        languages_map[current_language],
                        modes_dir,
                        modes_map[current_state])

    return audio_path


def signal_handler(sig, frame):
    GPIO.cleanup()
    sys.exit(0)

def forward_button_callback(channel):
    global s
    global
current_state,num_states,press_threshold,current_language,languages_map
    s = time.time()

    while(GPIO.input(FORWARD_BUTTON_GPIO) and time.time() - s <
threshold):
        time.sleep(0.05)

    else:
        print(time.time() - s)
        if(GPIO.input(FORWARD_BUTTON_GPIO) and time.time() - s >
threshold):
            img = picam2.capture_array("main")
```

```
            result =
perform_current_function(current_state,picam2,img,current_language)

            if(result):
                print(result)
            else:
                print("none")


            img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
            cv2.imwrite('test1.jpg',img)

            print("long press")


        elif(not GPIO.input(FORWARD_BUTTON_GPIO)
             and time.time() - s < threshold
             and time.time() - s > press_threshold):

            if(current_state == num_states):
                current_state = 1
            else:
                current_state += 1

            print('regular press')
            audio_file_to_be_played = get_audio_path(audios_path,
                                        languages_map,
                                        current_language,
                                        modes_dir,
                                        modes_map,
                                        current_state)+'.wav'
            wave_obj =
sa.WaveObject.from_wave_file(f'{audio_file_to_be_played}')
            play_obj = wave_obj.play()
            play_obj.wait_done()
            print(f'audio to be played : {audio_file_to_be_played}')
            print(f"state = {modes_map[current_state]}")
```

```
def reverse_button_callback(channel):
    global s
    global
current_state,num_states,press_threshold,current_language,languages_map
    s = time.time()

    while(GPIO.input(REVERSE_BUTTON_GPIO) and time.time() - s <
threshold):
        time.sleep(0.05)

    else:
        print(time.time() - s)
        if(GPIO.input(REVERSE_BUTTON_GPIO) and time.time() - s >
threshold):
            print("Voice Commands On")

        elif(not GPIO.input(REVERSE_BUTTON_GPIO) and time.time() - s
< threshold and time.time() - s > press_threshold):

            if(current_state == 1 or current_state==0):
                current_state = num_states
            else:
                current_state -= 1
            print('regular press')
            audio_file_to_be_played = get_audio_path(audios_path,
                                        languages_map,
                                        current_language,
                                        modes_dir,
                                        modes_map,
                                        current_state)+'.wav'
            wave_obj =
sa.WaveObject.from_wave_file(f'{audio_file_to_be_played}')
            play_obj = wave_obj.play()
            play_obj.wait_done()
            print(f'audio to be played : {audio_file_to_be_played}')

            print(f"state = {modes_map[current_state]}")

if __name__ == '__main__':
```

```
    picam2 = Picamera2()

    camera_config = picam2.create_still_configuration(main={"size":
(2048, 1536)},transform=Transform(vflip=True,hflip=True))

    picam2.configure(camera_config)
    # picam2.set_controls({"ExposureTime": 600000, "AnalogueGain":
1.0,"AwbEnable":True,"AwbMode":2,"Brightness" :0.05})
    picam2.set_controls({"ExposureTime": 300000,"AwbEnable":True,})

    picam2.start()
    time.sleep(2)


    GPIO.setmode(GPIO.BCM)
    GPIO.setup(FORWARD_BUTTON_GPIO, GPIO.IN,
pull_up_down=GPIO.PUD_UP)
    GPIO.setup(REVERSE_BUTTON_GPIO, GPIO.IN,
pull_up_down=GPIO.PUD_UP)
    wave_obj =
sa.WaveObject.from_wave_file("/home/pi/python/project/data/audios/ar/m
odes/welcome.wav")
    play_obj = wave_obj.play()
    play_obj.wait_done()
    GPIO.add_event_detect(FORWARD_BUTTON_GPIO, GPIO.RISING,
          callback=forward_button_callback, bouncetime=50)

    GPIO.add_event_detect(REVERSE_BUTTON_GPIO, GPIO.RISING,
          callback=reverse_button_callback, bouncetime=50)

    signal.signal(signal.SIGINT, signal_handler)
    signal.pause()
```

3 - Mobile App

GitHub Link:

https://github.com/MasameEh/eyes_app/tree/master

4 - FastAPI server

4.1 Endpoint (main.py)

```
from pathlib import Path
from fastapi import FastAPI, UploadFile

app = FastAPI()

# make a directory and it's ok if it exists, don't error
Path('media').mkdir(exist_ok=True)


@app.get('/')
async def home():
    return {
        "app": "is working"
    }


@app.post("/upload")
async def upload(file: UploadFile):
    # save the file
    with open(f'media/{file.filename}', 'wb') as f:
        data = await file.read()
        f.write(data)
```

```
    # return a message saying it's saved
    return {'done': True}
```

4.2 Api.service

```
[Unit]
Description=description about this service

[Service]
User=pi
WorkingDirectory=/home/pi/app
ExecStart=uvicorn main:app --host 0.0.0.0 --reload
Restart=always

[Install]
WantedBy=multi-user.target
```