IC 201P – Design Practicum

Project Name	Course	IC 201P
SmartGaze	Project Tagline	Power of Enhanced Vision
	Group Number	G5
	Team Members	Mantavya Gupta - B21015 Akarshan Kapoor - B21003 Bhumesh Gaur - B21040 Aayushi Thakur - B21028 Saqib Kales - B21128 Saurav Kumar Singh - B21264
	Supervising Mentors	• **Dr. Sayantan Sarkar** *sayantan@iitmandi.ac.in* • **Dr. Pratim Kundu** *pratim@iitmandi.ac.in*

Indian Institute of Technology Mandi

Certificate

This is to certify that the work contained in the project report entitled "***SmartGaze***", submitted by Group G5 to the Indian Institute of Technology Mandi, for the course IC 201P – Design Practicum, is a record of bonafide research works carried out by them under our direct supervision and guidance.

Date : 17 May, 2023

Signature and Date

Dr. Sayantan Sarkar

Dr. Pratim Kundu

Acknowledgements

First and foremost, we would like to thank our mentors, **Dr. Sayantan Sarkar** and **Dr. Pratim Kundu** for their invaluable guidance and feedback on improving our project and constantly giving ideas for improvement. We would also like to extend our appreciation towards our TA, **Mr. Vishal Gupta** who has provided us with his expertise and assistance. We are truly grateful towards them.

Abstract

The objective of this project is to design and develop a pair of smart glasses that can translate written language in real-time. By integrating optical character recognition (OCR) and using *tesseract* software, the glasses will be able to recognize and translate written text into another language, displaying the translation directly onto the lenses in real-time. This technology has the ability to easily commute or navigate in a country or state where one does not speak the local language is a significant advantage for travelers. This highlights the importance of translating essential elements such as signboards, which can greatly aid in the ease of communication and understanding.

Goals:

- *Develop an OCR system that can accurately recognize written text and convert it into machine-readable format.*
- *Integrate the translation output with the smart glasses lenses, displaying the translation in real-time.*
- *Ensure that the glasses are lightweight, comfortable, and aesthetically pleasing for everyday use.*
- *Test and refine the system to improve accuracy and optimize performance.*
- *Evaluate the glasses' effectiveness in real-world scenarios, assessing their potential impact on communication across linguistic barriers.*

Contents

List of Figures

- Code snippet for Working of PHP based local host server
- Code snippet for OLED Module
- Code snippet for Working of Translation Module
- Code snippet for Working of Facial Recognition
- Code snippet for Working of All Codes Together
- Code snippet for Automation using Crontabs

C. Mechanical Modeling

- 3D Model of Final Prototype and its Orthogonal View.
- Idea behind the Final Design of *Smart Gaze*.
- Individual parts inspection with dimensions using various views of the model
    - Right Side Box
    - Left Side Box
    - Back Side Box
    - Stems
    - Circular Tubes

Chapter 5 :

1. **Challenges**
    - Optical Solution For our Problem of 25cm near vision.

List of Tables

Abbreviations

- **OCR :** Optical Character Recognition

Chapter 1

Introduction

After brainstorming with our group and considering 30 top problems , we came to the major issues to be resolved by our product i.e., facial recognition, communication gap because of language barrier, and visual experiences to save valuable time.

SmartGaze :

The power of enhanced vision, it is basically a smart glass but not like the ones existing in the market. To collectively account for the problems like real - time translation(image, face recognition and visual data display to user), effective communication (language translator), enhanced vision we decided to make smart glasses with these advancements, which is quite different from the traditional methods of accessing and interacting with digital information .The problem solving scope of the smart glasses is significant it allows users to operate technology hands-free so that user can perform other tasks simultaneously, it can be used to provide immersive and interactive training / education experiences . It will also increase accessibility for people with disabilities (can provide real -time visual descriptions of the environment for people with visual impairments and will make communications effective and productive in industries and many other sectors by the use of inbuilt language translators.

Compared to other AR products like smartwatches, smart glasses can offer a more immersive experience because they often have a larger display and can be worn in front of the eyes. This can be especially helpful for applications like augmented reality, where a larger display can improve the capacity to overlay digital information on the real world and also offers hands free experience.

SmartGaze can be used for healthcare ,marketing ,transportation fields particularly using image recognition. Language translation features can be used in industries, Education & Training , Travel & Tourism, Entertainment etc. The OCR part can be used to read printed documentation , automatic number plate recognition etc.

Based on the unique contributions of each team member, the report has been cooperatively organized. The report states the main goal first, then the supporting

details. The report's structure was chosen to make it easier to read by starting with the Abstract and content page at the top of the document. The Introduction, Market Research, Product Design, Components Information, Software, and Algorithm are the following sections. In the end, the report seeks to bring all the findings together into one functional model. The sections titled "Conclusion," "References," and "Contributions" are found at the end of the report.

Chapter 2

Market Research

Market Analysis of different Smart Glasses:

Company	Features	Products	Price	Link
Smartprix	Visuals on screen.	Web cam, display oled screen.	*70,999*	SmartPrix
Mavigadget	Detailed navigation is visualized , takes pictures, and wirelessly communicates with the phone.	Touchpad , Web cam, visual display screen, speaker, microphone, phone	*64,499*	Mavigadget
Envision	Instant Text, Scan Text, Batch Scan, Call an Ally, Call Aira, Describe Scene, Find People, Find Objects, provides visual information around you	Web cam, oled screen,integrated speaker	*1,99,999*	Envision
Vuzix Smart Glasses	Providing access to location-aware information, data collection, communications with both audio and video, and more.	An auto-focus 8-megapixel camera, built-in stereo speakers, and advanced Vuzix voice control	*1,30,000*	Vuzix Blade
SmartGaze: **Our Project**	Data collection , Communication(language translator)	8-megapixel camera, Oled screen, Translation and Facial Recognition.	*30,000*	Will be updated in the near future.

The above table provides the information of existing Smart Glasses in the market as well as that of our project (SmartGaze). A brief comparison of our product with the existing ones is also given in terms of features.

SmartGaze in comparison to others:

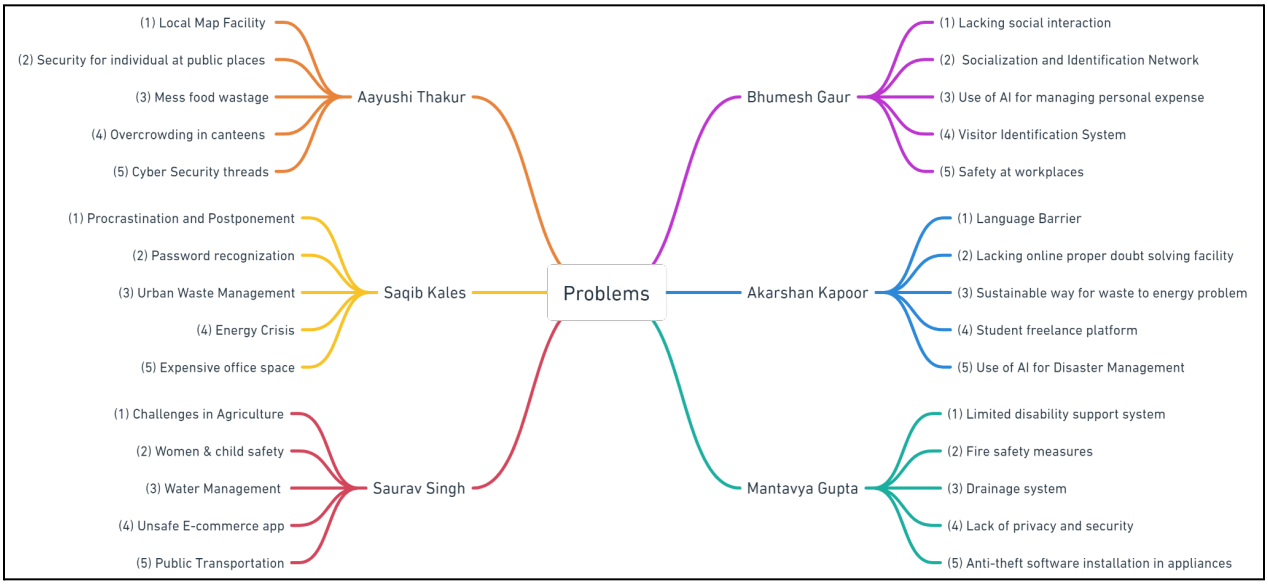
Most of the smart glasses present in the market provide limited functionality and are primarily used for taking photos, making calls and managing notifications. Few smart glasses are bulky and unattractive, which can be a major turn-off for many users, additionally they may not be comfortable to wear for extended periods of time. Many smart glasses are expensive, which can make them inaccessible to a lot of people.

SmartGaze will work as an aid in the market of smart glasses, our product is focused on the advanced features like, Language recognition and translation, image, Object and facial recognition to make it truly smart also the need of the user has also been considered by resolving design issues, making it slim, lightweight, attractive and comfortable. We have tried to make our product cost effective.

Chapter 3

Conceptual Design

Problems Classification Tree:



Brainstorming and Idea Generation:

Augmented Reality (AR) and Virtual Reality (VR) are rapidly evolving technologies that are transforming various industries and sectors, including healthcare, education and training, marketing and design, social media, and communication. These technologies provide innovative and transformative solutions to enhance user experiences, improve efficiency, and drive business growth. In this modern world, our group identified the following major problems:

1. Real-time translation
2. Breaking Communication barriers in a busy world.
3. Special Vision for humans to save time.

Inspired by online available solutions, we decided to create a handset device that can perform real-time translation and identify individuals using computer vision. Considering the concern of carrying a separate device, we decided to incorporate these functionalities into smart glasses, which can be worn like regular eyewear, providing a practical and convenient solution.

Problem Statement:

Our group has identified three major problems that we aim to address with our innovative smart glasses:

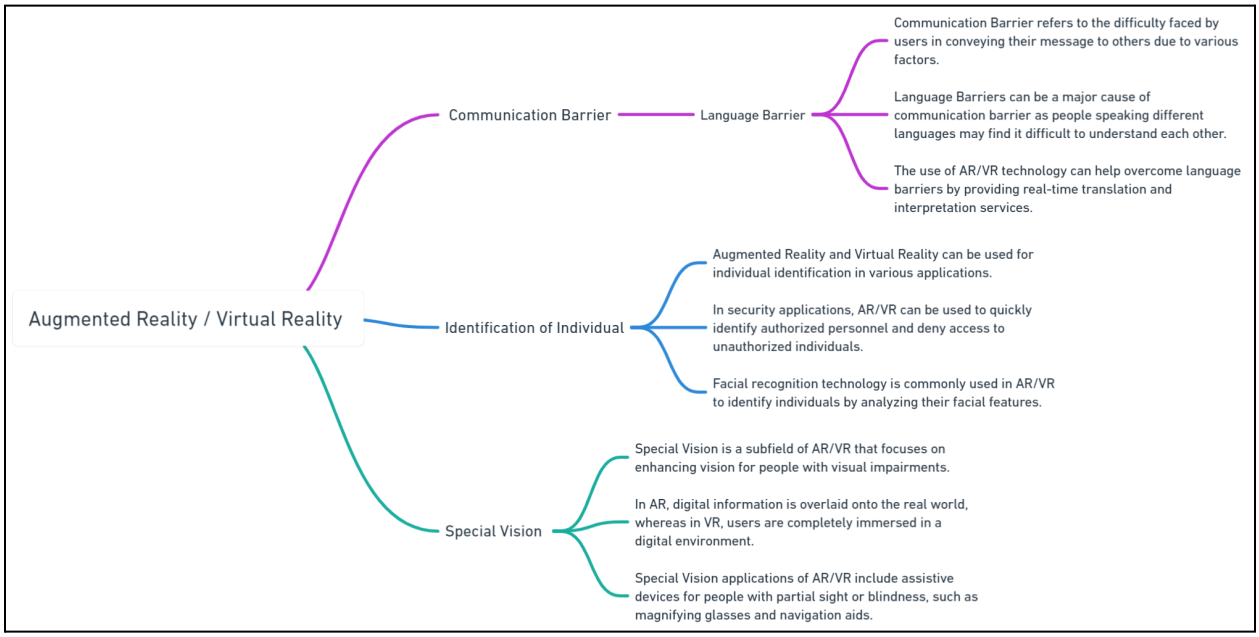
1. *Real-time translation:*
   Language barriers can hinder effective communication in various situations, such as travel, business meetings, and social interactions. Real-time translation can bridge these language gaps, enabling seamless communication between individuals speaking different languages.

2. *Breaking communication barriers in a busy world:*
   In today's fast-paced world, effective communication is crucial. However, factors such as background noise, distractions, and time constraints can hinder communication, leading to misunderstandings and misinterpretations. Our smart glasses aim to break these communication barriers by providing enhanced communication capabilities in various settings.

3. *Special vision for humans to save time:*

   Human vision has limitations, and we often miss important details or waste time searching for information. Our smart glasses will incorporate computer vision capabilities to provide users with special vision, enabling them to identify objects, individuals, and relevant information quickly and efficiently, saving time and enhancing productivity.

Solution:



To address the identified problems, our group has decided to create smart glasses that incorporate AR and computer vision technologies. These smart glasses will provide real-time translation, enhanced communication capabilities, and special vision functionalities to users. The key features of our smart glasses will include:

Real-time translation: Our smart glasses will have built-in language recognition and translation capabilities. Users can speak or listen to different languages, and the smart glasses will automatically translate the speech in real-time, enabling seamless communication with individuals speaking different languages.

Special vision: Our smart glasses will incorporate computer vision capabilities, such as image recognition, object detection, and facial recognition. Users can quickly identify objects, individuals, and relevant information by simply looking at them through the smart glasses, saving time and enhancing productivity.

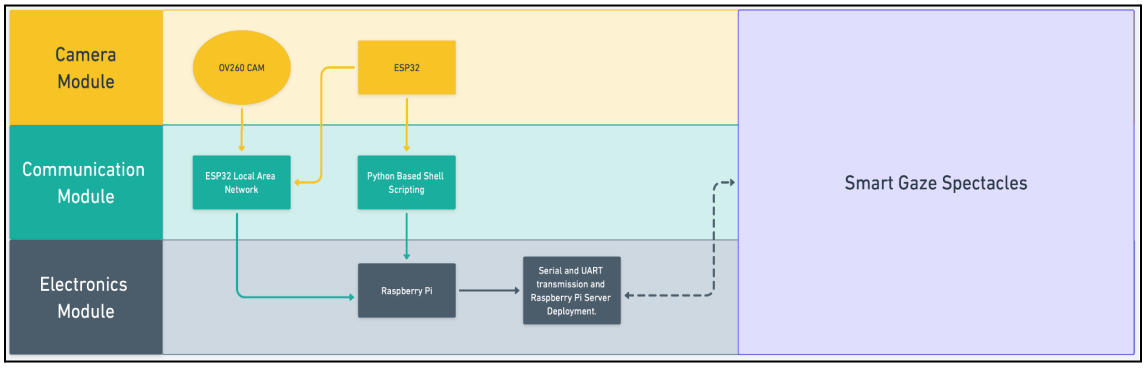
Stylish and comfortable design: Our smart glasses will have a sleek and stylish design, resembling regular eyewear to ensure comfort and ease of use. They will be lightweight, durable, and compatible with different face shapes and sizes, making them suitable for long-term wear.

Chapter 4

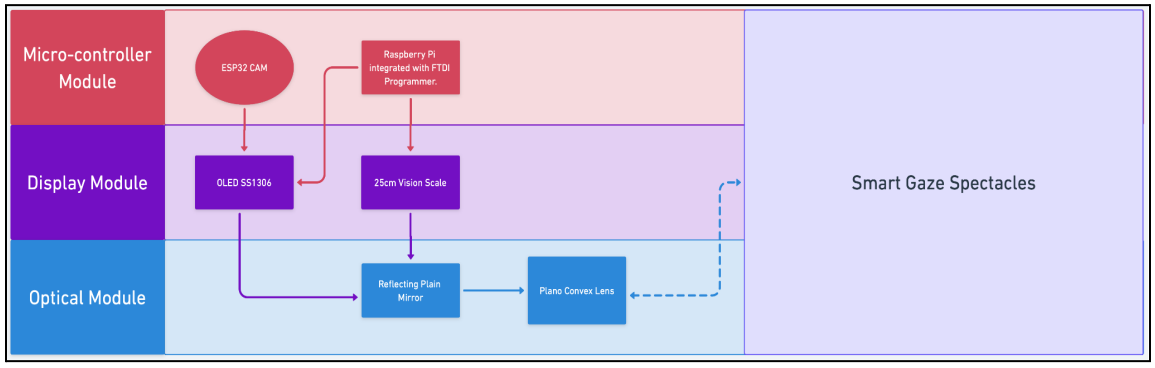
Embodiment and Detailed Design

Product Architecture:

- Highlight of Interior Electronic and Code Design:



- Highlight of Exterior Engineering and Physical Design:



- Working Concept:

1. The FTDI Board is connected to an ESP32 Cam, which is a powerful camera module that can capture images and videos. The camera is used for face recognition and language translation. The camera

captures the image of the text that needs to be translated, and the ESP32 Cam processes this image to extract the text. The text is then translated using an online translation API, and the translated text is displayed on the OLED display.

2. The OLED display is a compact display that consumes less power and has a high contrast ratio. The display is used to show the translated text and other valid information. The OLED display is connected to the Arduino Uno and is controlled by it. The display is small, but it is clear and easy to read.

3. The optics of the smart glasses are designed to take care of the minimum distance of vision using a Plano convex lens. The lens is a simple convex lens of focal length 100mm that is used to focus the light onto the retina of the eye. The lens is placed in front of the eye and is adjusted so that the minimum distance of vision is 25 cm. This ensures that the user can see the display clearly without any strain.

4. The final display is reflected into a film using a mirror. The mirror is placed at a 45-degree angle, and the display is reflected onto the film. This makes the display appear as if it is floating in front of the user's eyes. The film is a special film that is transparent and reflects the display. This final display is what the user sees and is used to display the translated text and other valid information from facial recognition.

- Feasibility of Design:

The above design of smart glasses is using an FTDI Board connected to an ESP32 Cam for language translation via camera and face recognition, an OLED display is also connected with Raspberry Pi 3 to show valid information, optics to take care of the minimum distance of vision using Plano convex lens, and a mirror to reflect the final display into a film is feasible due to several reasons.

1. Firstly, the components used in the design are easily available in the market. The Raspberry Pi 3 microcontroller board, ESP32 Cam,

OLED display, Plano convex lens, and mirror are all readily available components. This makes it easy to source and assemble the components needed for the smart glasses.

2. Secondly, the use of an ESP32 Cam for language translation via camera and face recognition is a well-established technology. The ESP32 Cam is a powerful camera module that can capture high-quality images and videos. It has built-in Wi-Fi and Bluetooth connectivity, making it easy to connect to the internet and access online translation APIs. Face recognition technology has also become increasingly popular and accurate in recent years, making it feasible to use for smart glasses.

3. Thirdly, the use of an OLED display to show valid information is an excellent choice due to its low power consumption and high contrast ratio. OLED displays are also lightweight, making them ideal for use in smart glasses.

4. Fourthly, the use of optics to take care of the minimum distance of vision using Plano convex lenses is a well-established technology used in eyewear. The Plano convex lens is a simple convex lens that can be easily adjusted to provide the minimum distance of vision required for the smart glasses.

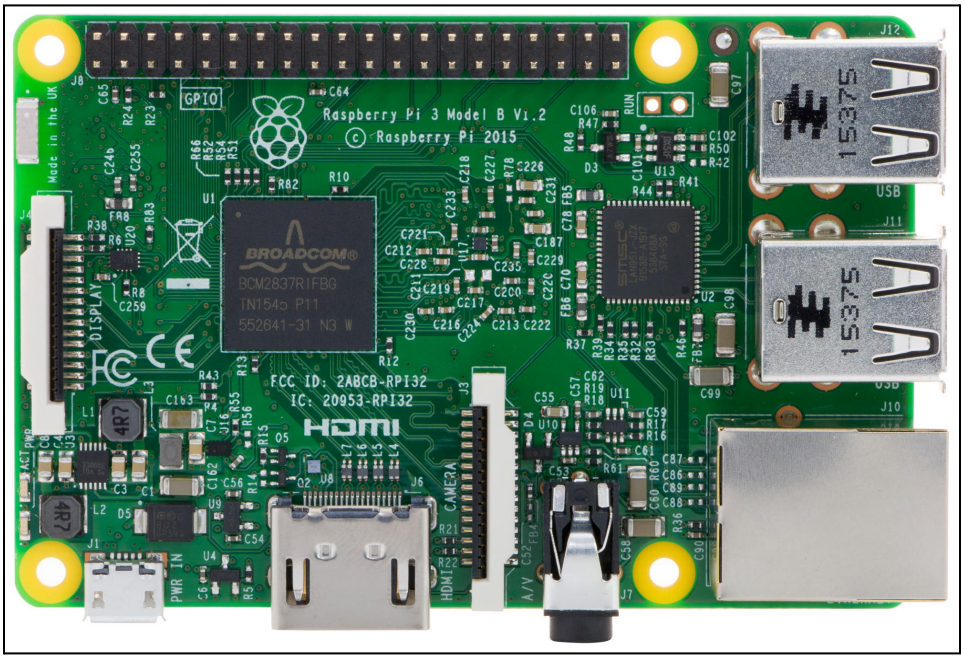
5. Lastly, the use of a mirror to reflect the final display into a film is a feasible design choice. The mirror is placed at a 45-degree angle, reflecting the display onto the film, which makes it appear as if it is floating in front of the user's eyes. This technology is widely used in optical devices such as binoculars and telescopes.

6. In conclusion, the above design of smart glasses is feasible due to the availability of components, established technologies used in language translation, face recognition, optics, and display technologies. Smart glasses have the potential to revolutionize the way we communicate and interact with the world around us.

System – Level Design:

As of now, the components utilized in this project are as follows:

- Raspberry Pi 3B [3]:



The Raspberry Pi 3 is a credit card-sized single-board computer that was released by the Raspberry Pi Foundation. It is designed to be a versatile and affordable platform for learning about and experimenting with computing and electronics. The Raspberry Pi 3 is commonly used for a wide range of projects, including home automation, robotics, media centers, game emulation, IoT (Internet of Things) applications, and learning programming and electronics. It provides an affordable and accessible platform for enthusiasts, hobbyists, and students to experiment and create their own projects.

1. 1.2GHz 64-bit quad-core ARM Cortex-A53 processor.
2. 1GB of RAM.
3. Built-in Wi-Fi (802.11n) and Bluetooth 4.2 support.
4. Four USB 2.0 ports.
5. HDMI port for display connectivity.
6. Ethernet port for wired networking.

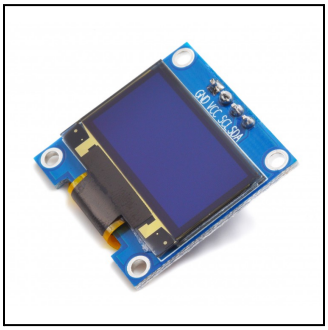
7. 3.5mm audio jack for audio output.
8. MicroSD card slot for storage.
9. Compatible with various operating systems like Raspbian, Ubuntu, and Windows 10 IoT Core.
10. GPIO (General-Purpose Input/Output) pins for physical computing and interfacing with external hardware.

- ESP32 CAM WiFi Module Bluetooth with OV2640 Camera Module:

The ESP32 CAM WiFi Module Bluetooth with OV2640 Camera Module 2MP For Face Recognition has a very competitive small-size camera module that can operate independently as a minimum system with a footprint of only 40 x 27 mm; a deep sleep current of up to 6mA and is widely used in various IoT applications. It is suitable for home smart devices, industrial wireless control, wireless monitoring, and other IoT applications.

1. OV2640 2MP camera module
2. Wi-Fi connectivity
3. Bluetooth connectivity
4. Support for JPEG encoding
5. Suitable for capturing Images and video
6. Ideal for IoT and wireless communication applications

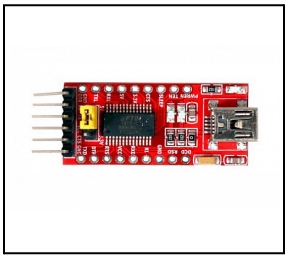
- OLED Display Module SSD1306:



SSD1306 is a CMOS OLED driver with a controller for an OLED dot-matrix graphic display system. Due to use of the SSD1306 driver, the number of external components required and power consumption has reduced. An OLED display is used for displaying text, images and various patterns.

1. OLED technology for high contrast ratios, deep blacks, and wide viewing angles.
2. Display resolution options of 128x64 pixels or 128x32 pixels.
3. Supports I2C, SPI, and parallel communication interfaces.
4. Integrated SSD1306 controller for efficient display operations.
5. Low power consumption.
6. Suitable for detailed graphics and text displays. Compact and lightweight form factor.
7. Compatible with various microcontrollers and development boards.
8. Wide operating temperature range.
9. Can be used in a variety of applications such as wearables, IoT devices, and embedded systems.

- FTDI Programmer



An FTDI programmer is a device that uses the FTDI (Future Technology Devices International) chip to enable communication between a computer and electronic devices that use serial interfaces, such as microcontrollers, Arduino boards, and other programmable devices.

1. Utilizes an FTDI chip for serial communication.
2. Connects to the computer via USB.
3. Enables bidirectional data transfer between the computer and the target device.
4. Requires driver installation on the computer.
5. Used for programming and debugging microcontrollers and programmable devices.
6. Compatible with various operating systems and programming environments.
7. Widely used in electronics prototyping, embedded systems development, robotics, and IoT projects.

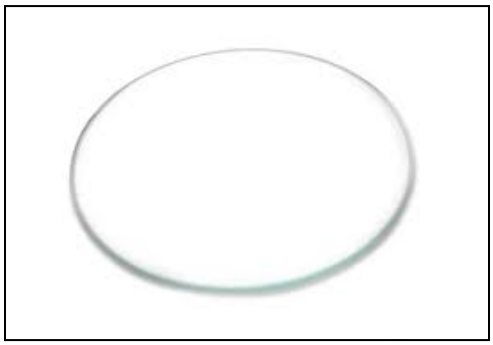
- Lithium Ion Battery

A lithium-ion or Li-ion battery is a type of rechargeable battery which uses the reversible reduction of lithium ions to store energy. The anode (negative electrode) of a conventional lithium-ion cell is typically graphite made from

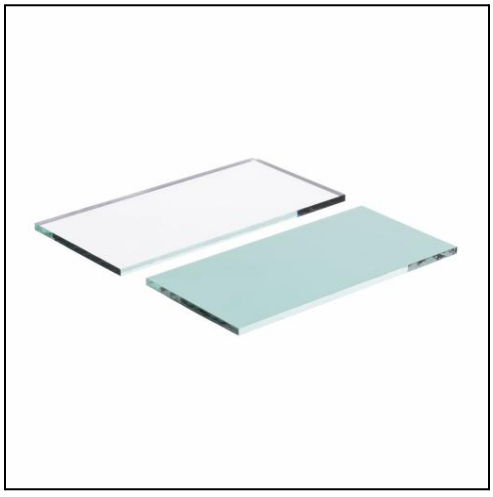
carbon. The cathode (positive electrode) is typically a metal oxide. The electrolyte is typically a lithium salt in an organic solvent.

8. High energy density
9. Long cycle life
10. Fast charging
11. Lightweight and compact
12. Widely used in portable electronics and electric vehicles
13. Low self-discharge rate
14. No memory effects
15. Relatively low environmental impact
16. Requires protection circuitry to prevent overcharging and overheating
17. Sensitive to high temperatures and can degrade if exposed to extreme conditions.

- Plano Convex Lens with Focal Length of 100 mm



- Reflecting Mirror



Design Configuration:

Component	Dimensions	Shape
Raspberry Pi 3	• Length: 85.60 mm (3.37 inches) • Width: 56.5 mm (2.22 inches) • Height: 17 mm (0.67 inches)	Rectangular
ESP32 CAM WiFi Module Bluetooth with OV2640 Camera Module	• Length: 40 mm (1.57 inches) • Width: 27 mm (1.06 inches) • Height: 12 mm (0.47 inches)	Rectangular
FTDI Programmer	• Length: 40 mm (1.57 inches) • Width: 20 mm (0.79 inches) • Height: 10 mm (0.39 inches)	Rectangular
OLED Display Module SSD1306	• Length: 27 mm (1.96 inches) • Width: 27 mm (1.06 inches)	Square
Plano Convex Lens	• Diameter: 5 cm (1.06 inches) • Focal Length: 100 mm (3.93 inches)	Circular
Lithium Ion Battery	• Diameter: 18mm (0.70 inches) • Length: 65mm (2.56 inches)	Cylindrical

Detailed Design:

- **Electrical Assembly:**

To connect an FTDI Programmer to an ESP32 Cam, we follow the steps below:

Have the necessary libraries installed in your Arduino IDE to program the ESP32 Cam. The libraries required for the ESP32 Cam include the ESP32, ESPAsyncWebServer, and ESPAsyncTCP libraries. Further board URLs that identify the ESP32 Wrover Module and AI-Thinker Module also need to be added.

Connect the ESP32 Cam to the FTDI [2] as follows:

1) Connect the GND pin on the ESP32 Cam to a GND pin on the FTDI.

2) Connect the 5V pin on the ESP32 Cam to the 5V pin on the FTDI.

3) Connect the U0R pin on the ESP32 Cam to the TXD pin on the FTDI.

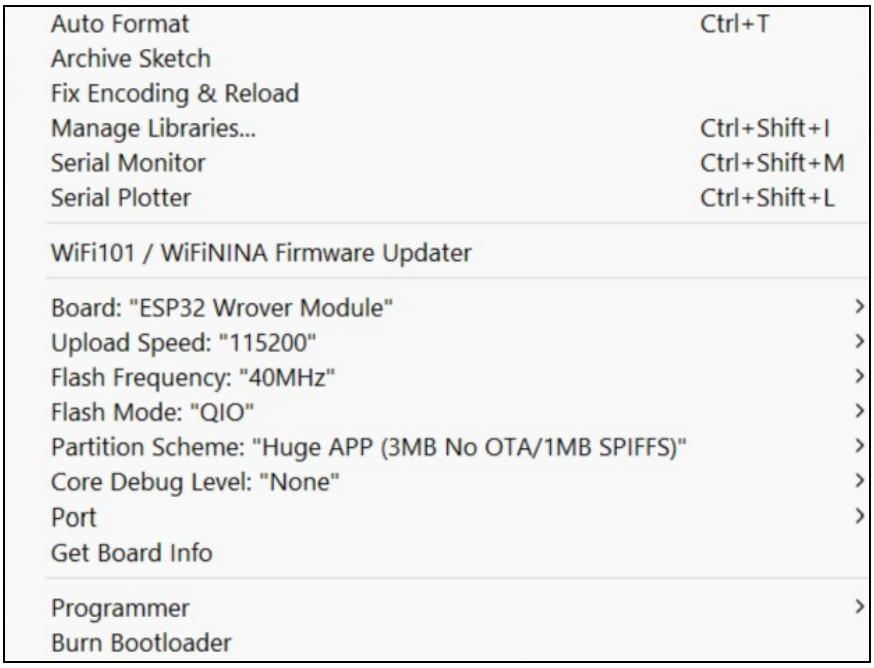
4) Connect the U0T pin on the ESP32 Cam to the TXD pin on the FTDI.

5) Connect the IO0 pin on the ESP32 Cam to it's own GND (This is done to put the ESP32 Cam into flashing mode)

<u>Note:</u> The ESP32 Cam uses 3.3V logic, while the FTDI uses 5V logic. Hence, it is important to use a logic level converter to avoid damaging the ESP32 Cam. This is self implemented in ESP32 Cam.

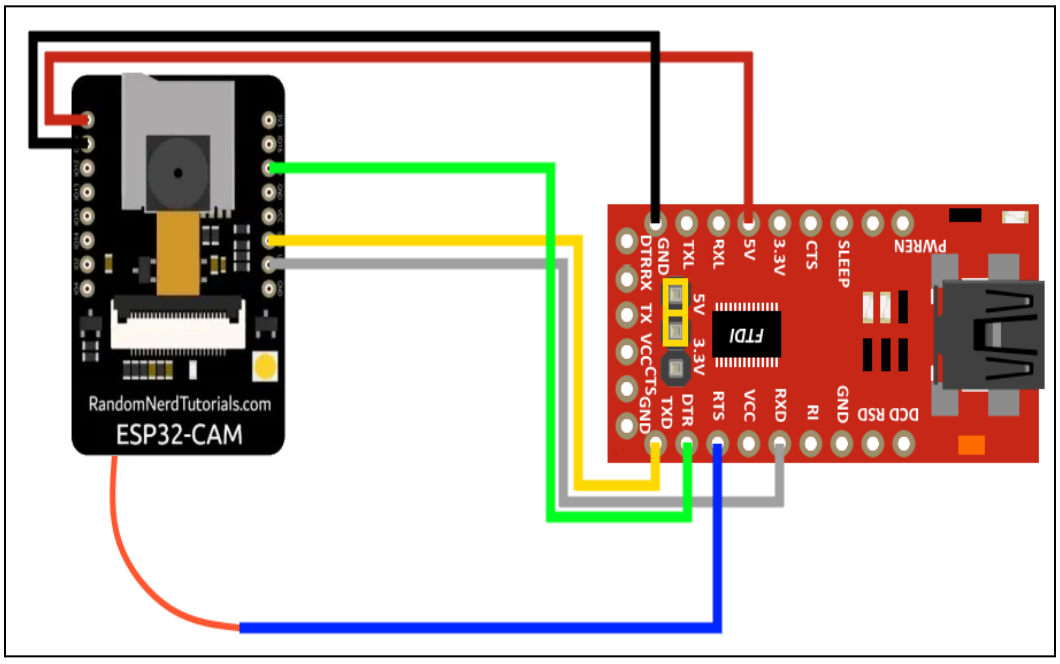
Once the connections have been made, open the Arduino IDE and select the correct board and port settings. For the ESP32 Cam, select the "ESP32 Wrover Module" board from the boards menu, and select the correct port from the tools menu.

Other parameters should be set as shown below:

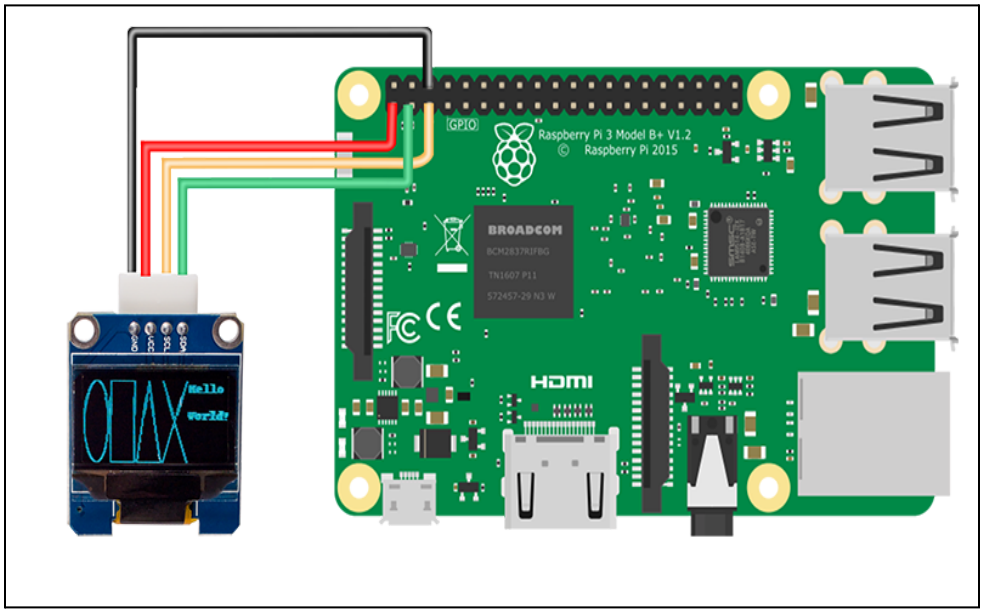


The following diagrams are to be followed for proper wire connections:

Connecting ESP-32 to FTDI Board:



Connecting SSD 1306 OLED Display to Raspberry Pi 3B



Connect the OLED Display to the Raspberry Pi 3B module [1] as follows:

- 3V3 Power (Pin 1) ---------- VCC (Power) on OLED
- Ground (Pin 6) ---------- GND (Ground) on OLED
- SDA (I2C Data) (Pin 3) ---------- SDA (Data) on OLED
- CL (I2C Clock) (Pin 5) ---------- SCL (Clock) on OLED

- **Software Compilation:**

- Working of the camera module [3]:

1) *Working of PHP based local host server:* A snippet of the same is as follows:

ImageCapture.ino

```
esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    ESP.restart();
  }

  server.on("/", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send_P(200, "text/html", index_html);
  });

  server.on("/capture", HTTP_GET, [](AsyncWebServerRequest * request) {
    new_photo = true;
    request->send_P(200, "text/plain", "Capturing Photo using ESP32-CAM");
  });

  server.on("/saved-photo", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send(SPIFFS, photo_path, "image/jpg", false);
  });


  server.begin();
```

1. This code sets up an ESP32-CAM as a web server to capture and display photos. It first includes several libraries, including the WiFi library, the ESP Camera library, and the AsyncWebServer library. It defines the SSID and password for the WiFi network and creates an instance of the AsyncWebServer class on port 80.
2. Next, it sets a boolean variable called new_photo to false and defines the file path for the captured photo. It then defines the pins for the OV2640 camera module and creates a string containing HTML code to be displayed on the web page.
3. In the setup() function, it initializes the serial port, connects to the WiFi network, and mounts the SPIFFS file system. It then turns off the 'brownout detector', initializes the OV2640 camera module with the defined pin configuration, and

checks the result of the initialization.

4. The main part of the code is the HTTP request handler that responds to a GET request for the /capture URL. When the request is received, it sets the new_photo flag to true, captures a new photo, saves it to SPIFFS, and returns the path to the saved photo.
5. The HTML code defined earlier is then sent to the client's web browser when the client requests the root URL, "/", which displays the web page. When the client clicks the "CAPTURE PHOTO" button, a JavaScript function sends a GET request to the /capture URL. When a response is received, the JavaScript updates the image tag to display the new photo.
6. There are also two other buttons on the web page: "ROTATE PHOTO" and "REFRESH PAGE". The "ROTATE PHOTO" button calls a JavaScript function that rotates the displayed photo by 90 degrees and updates the CSS styling of the container div based on the current rotation angle. The "REFRESH PAGE" button simply reloads the current page.

The software compilation is essentially divided into 2 parts :

1) Working of the OLED Display [4]: A snippet of the same is as follows:

oled_display.py

```
import Adafruit_SSD1306
import time
from PIL import Image,ImageDraw, ImageFont

disp = Adafruit_SSD1306.SSD1306_128_64(rst=None)
disp.begin()
disp.clear()
disp.display()

image = Image.new('1', (disp.width, disp.height))

draw = ImageDraw.Draw(image)

font = ImageFont.load_default()

with open("/home/ak/output.txt","r") as file:
        lines=file.readlines()

disp.clear()
disp.display()

y = 0
for line in lines:
        draw.text((0, y), line.strip(), font = font, fill=255)
        y +=8

disp.image(image)
disp.display()
```

Explanation of the code:

1. The code is using the Adafruit_SSD1306 library to control an SSD1306-based OLED display with a resolution of 128x64 pixels. The display is connected to the Raspberry Pi or another compatible board.
2. The code initializes the display, clears any existing content, and creates a new blank image using the PIL (Python Imaging Library) module. The image is created in a 1-bit monochrome format suitable for the OLED display.
3. It then opens a file named "output.txt" located in the "/home/ak/" directory and reads its contents. Each line of the file is stored in the lines variable.
4. The code iterates through each line in lines, and for each line, it uses the draw.text() method to render the text on the image. The strip() method removes any leading or trailing whitespace from the line. The font argument specifies the font to be used, and fill=255 sets the text color to white. The y variable keeps track of the vertical position of each line, and it increments by 8 pixels for each line to ensure proper line spacing. Finally, the updated image is displayed on the OLED screen using the disp.image() and disp.display() methods.

2) Working of the Translation Part [4]:

final_translation.py

```python
import requests
from bs4 import BeautifulSoup
from PIL import Image
from io import BytesIO
from urllib.parse import urljoin
import pytesseract
from PIL import Image
from translate import Translator


# Code for HTML Parsing and Storing Images : ##1

# Fetch the HTML content from the webpage
url = 'http://192.168.58.55/'
response = requests.get(url)
html_content = response.text

# Parse the HTML content using BeautifulSoup
soup = BeautifulSoup(html_content, 'html.parser')

# Find the image element in the HTML using its id
image_element = soup.find('img', id='photo')

# Extract the relative image source URL from the image element
relative_image_url = image_element['src']

# Construct the full image URL by combining with the base URL of the webpage
image_url = urljoin(url, relative_image_url)

# Fetch the image binary data from the image URL
image_response = requests.get(image_url)
image_content = BytesIO(image_response.content)

# Open the image using PIL
image = Image.open(image_content)

# Run Tesseract OCR on the image
text = pytesseract.image_to_string(image)

# Translate the input text from English to German
translator = Translator(to_lang="de")
translated_text = translator.translate(text)

# Open the output file in write mode and save the translated text
with open('final.txt', 'w') as output_file:
    output_file.write(translated_text)
```

Explanation of the code:

1. The code utilizes the requests library to make an HTTP request to a specified URL (http://192.168.58.55/) and fetch the HTML content of the webpage. It stores the response in the response variable.
2. Using the BeautifulSoup library, the HTML content is parsed and stored in the soup variable. This allows for easy traversal and extraction of specific elements from the HTML structure.
3. By using the find method on the soup object, the code locates an <img> element with the specified id attribute ('photo'). This represents an image element within the HTML.
4. The code extracts the relative URL of the image from the src attribute of the image element. It then uses urljoin from the urllib.parse module to construct the complete URL of the image by combining it with the base URL of the webpage.
5. With the complete image URL, another HTTP request is made to fetch the binary content of the image. The response content is stored in image_content, which is a BytesIO object representing the image data.
6. The code utilizes the pytesseract library to perform Optical Character Recognition (OCR) on the image. It uses the image_to_string function to extract text from the image. The resulting text is then translated from English to German using the translate library and saved to an output file named "final.txt".

(3) *Working of Facial Recognition [4]:* A snippet of the same is as follows:

facial_recognition.py

```python
import face_recognition as fr
import os
import pickle

def encode_faces(folder, use_cache=True):
    cache_path = os.path.join(folder, 'encoding_cache.pkl')
    if use_cache and os.path.exists(cache_path):
        with open(cache_path, 'rb') as f:
            return pickle.load(f)
    else:
        def generate_encodings():
            for filename in os.listdir(folder):
                known_image = fr.load_image_file(os.path.join(folder, filename))
                known_encoding = fr.face_encodings(known_image)[0]
                yield known_encoding, filename

        list_people_encoding = list(generate_encodings())

        if use_cache:
            with open(cache_path, 'wb') as f:
                pickle.dump(list_people_encoding, f)
        return list_people_encoding

def find_target_face():
    face_locations = fr.face_locations(target_image)
    print("PART 2  Proceeding")
    target_encodings = [target_encoding]
    for person in encode_faces('people/'):
        encoded_face = person[0]
        filename = person[1]
        is_target_face = fr.compare_faces(target_encodings, encoded_face, tolerance=0.6)
        print(f'{is_target_face} {filename}')

target_image = fr.load_image_file("image.jpg")
target_encoding = fr.face_encodings(target_image)[0]
print("Starting....")

print("Part 1 Done")

find_target_face()
```

Explanation of the code:

1. The code utilizes the face_recognition library, imported as fr, which provides functionalities for facial recognition tasks.

2. The encode_faces function is defined to generate face encodings for images in a specified folder. It checks for a cache file and returns the encodings if found, otherwise iterates through the image files, loads each image, and generates face encodings using fr.face_encodings.

3. The function generate_encodings is nested within encode_faces and yields each face encoding along with the corresponding filename.

4. The find_target_face function is defined to locate a target face in an image. It uses fr.face_locations to find face locations in the target image and stores the results in face_locations.

5. The target encoding is obtained by applying fr.face_encodings to the target_image and storing the result in target_encoding.

6. A loop in find_target_face iterates over the face encodings generated by encode_faces('people/'). Each person's encoding and filename are retrieved.

7. fr.compare_faces is used to compare the target_encoding with each encoded_face from the loop, using a tolerance of 0.6. The comparison determines if the target face matches the current encoded face, and the result is stored in is_target_face.

8. The code prints the result (True or False) of the comparison along with the corresponding filename for each person in the 'people/' folder.

9. The code loads an image from the file "image.jpg" using fr.load_image_file and obtains the face encoding for the target image using fr.face_encodings.

c) *Working of the final compilation [4]* : A snippet of the same is as follows:

final_compilation.py

```
import requests
from bs4 import BeautifulSoup
from PIL import Image
from io import BytesIO
from urllib.parse import urljoin
import pytesseract
from PIL import Image
from translate import Translator
import face_recognition as fr
import os
import pickle

# Function definitions:
def encode_faces(folder, use_cache=True):
    cache_path = os.path.join(folder, 'encoding_cache.pkl')
    if use_cache and os.path.exists(cache_path):
        with open(cache_path, 'rb') as f:
            return pickle.load(f)
    else:
        def generate_encodings():
            for filename in os.listdir(folder):
                known_image = fr.load_image_file(os.path.join(folder, filename))
                known_encoding = fr.face_encodings(known_image)[0]
                yield known_encoding, filename

        list_people_encoding = list(generate_encodings())

        if use_cache:
            with open(cache_path, 'wb') as f:
                pickle.dump(list_people_encoding, f)
        return list_people_encoding

def find_target_face():
    face_locations = fr.face_locations(target_image)
    print("PART 2  Proceeding")
    target_encodings = [target_encoding]
    out=[]
    for person in encode_faces('people/'):
        encoded_face = person[0]
        filename = person[1]
        is_target_face = fr.compare_faces(target_encodings, encoded_face, tolerance=0.6)
        out.append(f'{is_target_face} {filename}')
    print(out)
    with open('output.txt', 'w') as output_file:
         for e in out:
          output_file.write("%s\n" %e)
```

final_compilation.py

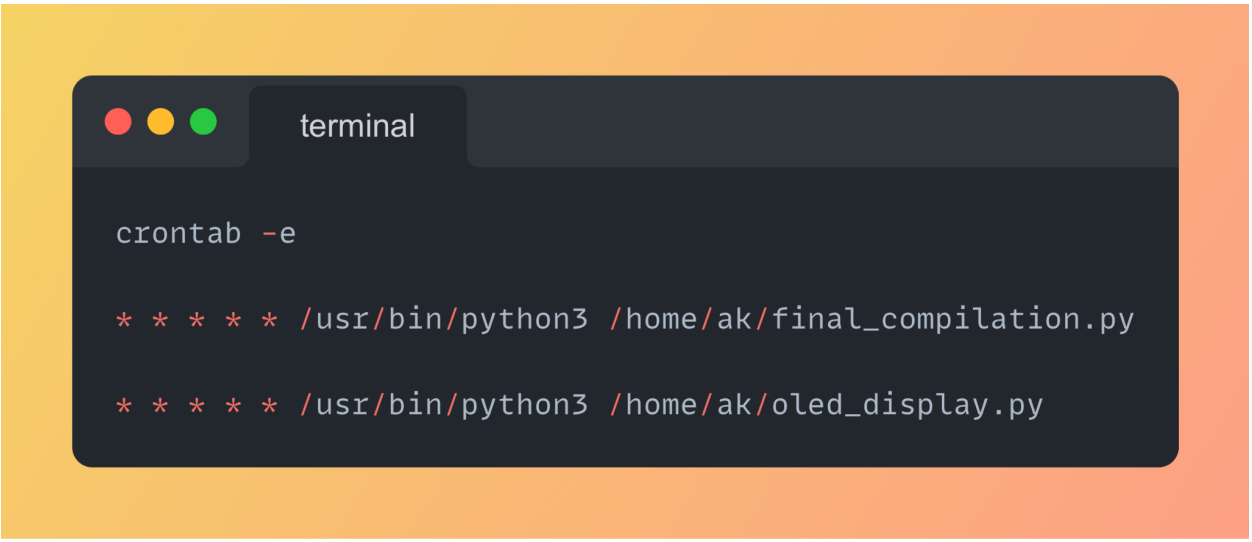
```python
# Code for HTML Parsing and Storing Images : ##1

# Fetch the HTML content from the webpage
url = 'http://192.168.48.55/'
response = requests.get(url)
html_content = response.text

# Parse the HTML content using BeautifulSoup
soup = BeautifulSoup(html_content, 'html.parser')

# Find the image element in the HTML using its id
image_element = soup.find('img', id='photo')

# Extract the relative image source URL from the image element
relative_image_url = image_element['src']

# Construct the full image URL by combining with the base URL of the webpage
image_url = urljoin(url, relative_image_url)

# Fetch the image binary data from the image URL
image_response = requests.get(image_url)
image_content = BytesIO(image_response.content)

# Open the image using PIL
image = Image.open(image_content)

image.save('image.jpg')
print("Image Saved Succesfully.")
# Run Tesseract OCR on the image
text = pytesseract.image_to_string(image)

if(text.isspace() or text==""):
    print("Text Not Found..... Starting Face Recognition")
    target_image = fr.load_image_file("image.jpg")
    try:
        target_encoding = fr.face_encodings(target_image)[0]
        print("Starting....")
        print("Part 1 Done")
        find_target_face()
    except Exception as e:
         output = "Image Unclear.....Try Again!!"
         print(output)
         with open('output.txt', 'w') as output_file:
            output_file.write(output)
else:
    print("Text Found....Starting Translation")
    # Translate the input text from English to German
    translator = Translator(to_lang="de")
    translated_text = translator.translate(text)
    print(translated_text)
    # Open the output file in write mode and save the translated text
    with open('output.txt', 'w') as output_file:
        output_file.write(translated_text)
```

Explanation of the code :

1. The code imports necessary libraries, including requests, BeautifulSoup, PIL, BytesIO, urljoin, pytesseract, face_recognition, os, and pickle, to perform various tasks related to web scraping, image processing, OCR, translation, and face recognition.
2. The code defines the encode_faces function to generate face encodings for images in a specified folder. It checks for a cache file and returns the encodings if found; otherwise, it iterates through the image files, loads each image, and generates face encodings using fr.face_encodings.
3. The find_target_face function is defined to locate a target face in an image. It uses fr.face_locations to find face locations in the target image and stores the results in face_locations.
4. The HTML content is fetched from a webpage specified by the URL using requests.get. The content is then parsed using BeautifulSoup and stored in the soup variable.
5. The code finds the image element in the HTML using its id, photo, using soup.find. The relative image source URL is extracted from the src attribute of the image element.
6. The full image URL is constructed by combining the relative URL with the base URL of the webpage using urljoin.
7. The image binary data is fetched from the image URL using requests.get and stored in image_content, a BytesIO object.
8. The image is opened using PIL (Image.open) and saved as "image.jpg".
9. OCR is performed on the image using pytesseract.image_to_string to extract the text.
10. If no text is found (empty or only whitespace), the code proceeds to face recognition. It attempts to load the target image, perform face encoding using fr.face_encodings, and then calls find_target_face to identify the target face among the known faces.
11. If text is found, the code proceeds to translation. The text is translated from English to German using the translate library and saved in translated_text.

12. The translated text is written to the "output.txt" file.
13. The code includes error handling in case there are issues with face recognition or the image being unclear.
14. Various messages are printed to provide information about the progress and status of the operations being performed.
15. The output, whether it's the face recognition results or translated text, is also written to the "output.txt" file.
16. The code combines HTML parsing, image processing, OCR, translation, and face recognition to extract information from a webpage, perform text or face recognition based on the content, and save the results to a file.

c) *Automation using Crontabs [5] :* A snippet of the same is as follows[5]:



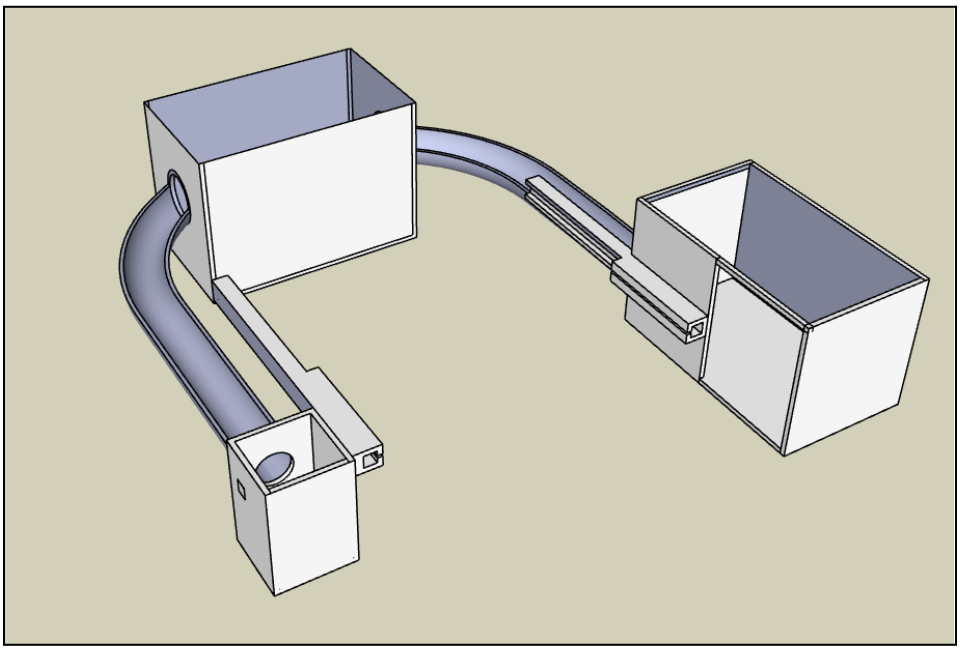
Explanation of the code :

1. The code sets up a crontab on the server and thus automates the execution of final_compilation.py and oled_display.py on the Raspberry Pi Server.

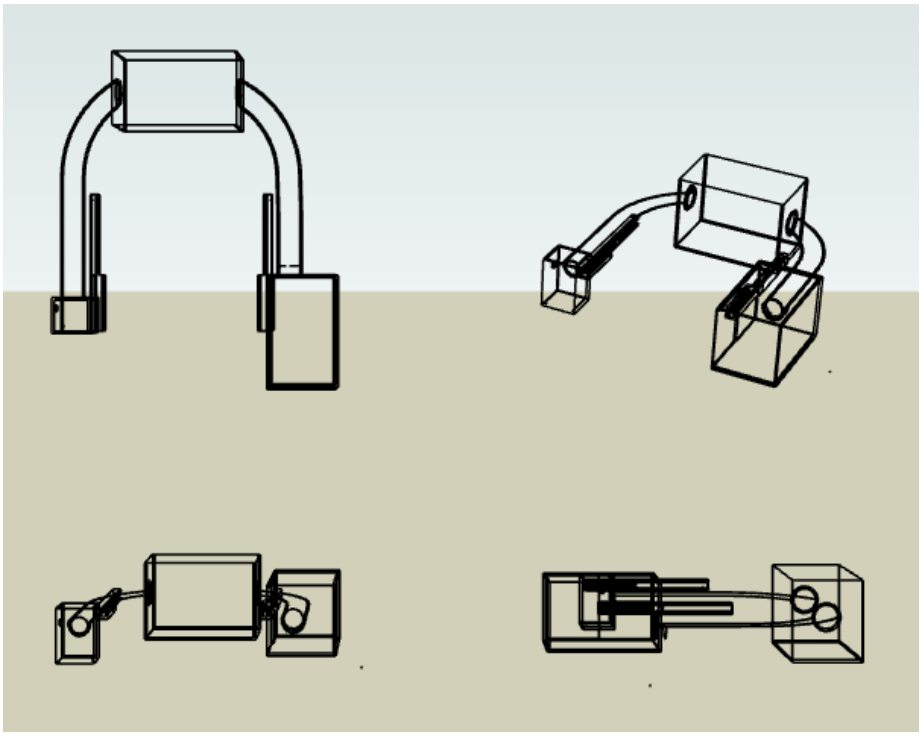
- **Mechanical Modeling :**

➢ **3D Model of Final Prototype:**

1. Box in the right side contains an ESP32 module along with a push button to capture the desired image.
2. Left side box contains an OLED screen along with a Plano convex lens and mirror.
3. Box at the back has Raspberry Pi , FTDI and a power bank inside it.



4. Two circular tube shaped paths have been provided from the back side box to both left and right side cases in order to achieve desired circuitry.
5. Two stems have been provided to hold the users spectacles and to obtain a stable configuration of all the parts and components in various aspects.



➢ **Idea behind the Final Design of *Smart Gaze* :**

In order to come up with a design for *Smart Gaze* we majorly focused on three sectors first i.e. **Comfort, Compact design** and **Cost effectiveness.**

Firstly , to make it comfortable all the components are placed such that the user doesn't feel irritated by any kind of blockage and all the circuitries have been enclosed in a circular tube shaped thing so that the user won't come in direct contact with wires and feel uncomfortable. All the components are placed inside the boxes which have significant distance from users' face , it also can be seen in the case of the Left side box, unlike the right one it has been given maximum length in vertically upward direction in order to avoid any kind of contact with users' face . The placement of various components is done in such a manner that weight is uniformly distributed all over the spectacles and there is no instability.

Secondly, we have designed the prototype in a very compact manner by inserting each possible component in boxes and reducing the circuit length as well to ensure less weight to be carried by the user.

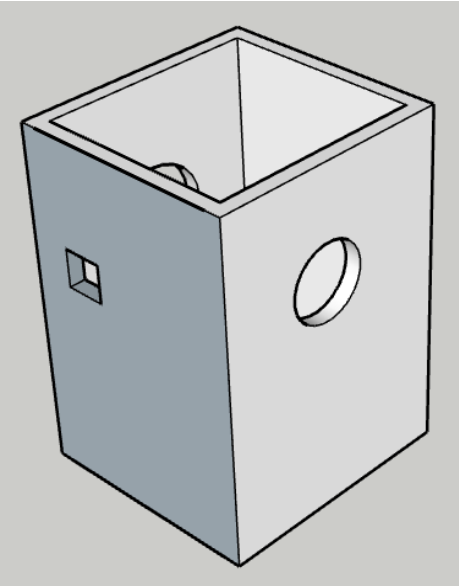
For cost efficiency, we have 3D printed our each part which is contributing to hold the various components with a density at which it can sustain desired loads and boxes have been provided lids at the top so that whenever there is a need of checking or modifying circuitry it could be done without damaging the components and parts as well. This will reduce any kind of cost to repairing and rebuilding the parts.

➢ **Individual parts inspection with dimensions using various views of the model :**

1. Right Side Box : **31*30*45 mm (L*B*H mm)**

   It has provided sufficient dimensions to hold an ESP32 **(40*26*22mm)** module inside it with connecting wires.

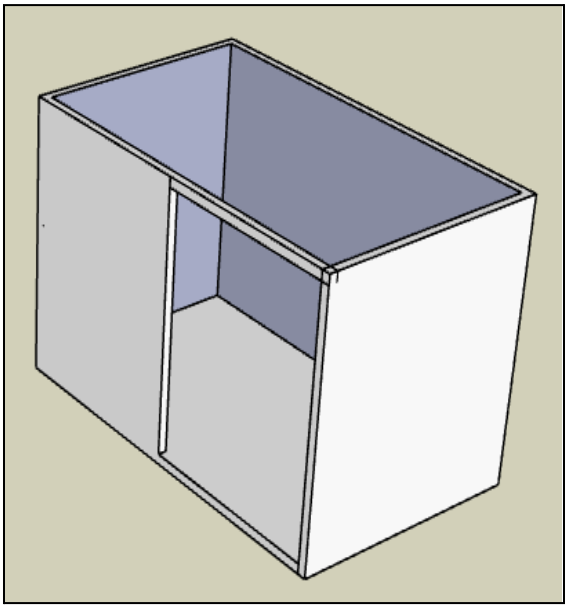
   A square hole **(5*5mm)** is also given for placing the push button which is directly being connected to the module therein.



2. Left side Box : **55*91*56mm**

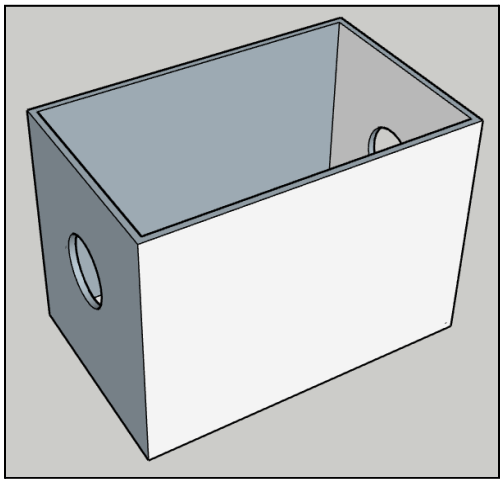
   It can easily accommodate an Oled (28*26*15) plano convex lens (dia-5mm) and a mirror (6*5mm).

It has provided also provided with a (44*52mm) open space to allow the light rays to pass through and strike the film present adjacent to the box in front of eyes at an angle.



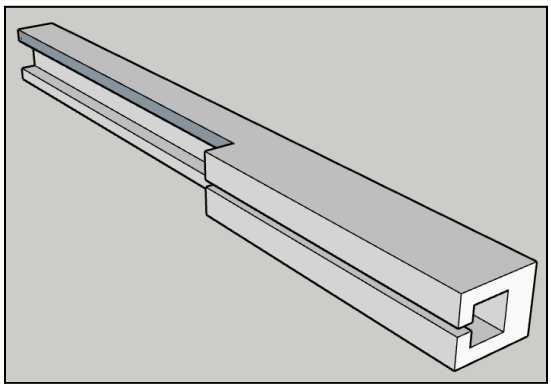
3. Back side Box : **99*63*69 mm**

   This box is having Raspberry pi and power bank both with dimensions (25*61*91mm) both are placed in such a way that longest dimension is horizontal and it also have FTDI (41*22*15mm). Holes have been provided for wires to run throughout the circuit.



4. Stems : **110*9*11mm**

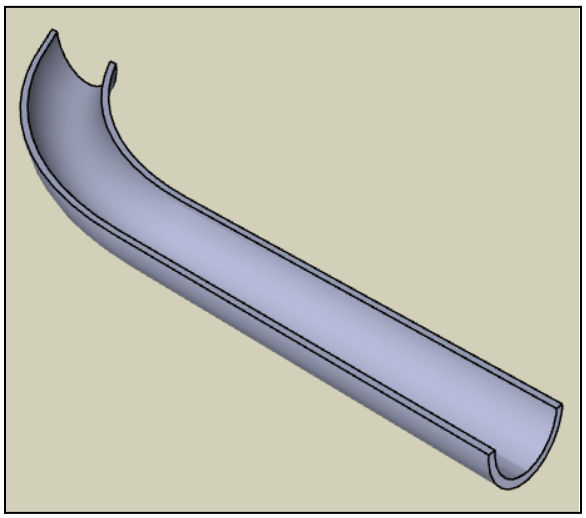
   Left stem is connected to the left box and so is the right stem .These stems have been provided to hold the spectacles stem and is the primary point of connection between the smart gaze and users' spectacles. It has small projections from top and bottom in order to make grip.



5. Circular Tubes : **Dia 10mm , Thickness 2mm**

   These tubes are running from left box and right box to the back side box in order to complete desired circuitry and compact the design. It has given optimum thickness as well diameter to attain even the thickest wire which is being used in circuit formation.

   It has been made in two parts, top and bottom. Top one will work as the lid and the bottom one will always remain attached to the base configurations.



Design of all the different parts is done taking into account the necessity of connection , compactness and weight distribution. We also tried to make it as slim as possible , but somewhere it isn’t possible to reduce dimensions and make it attractive , small and slim design wise.

Chapter 5

Fabrication and Assembly

Bill of Materials(BOM):

Bill of Materials *for*

SmartGaze- Power of Enhanced Vision

Component Name	Usage	Price/Quantity
1. Raspberry Pi 3B	Microcontroller with 1GB RAM used for setting up the server and automation protocol.	Rs. 2600 x 1
2. ESP32 Camera Module	A 2 MegaPixels Camera that will be used for live-video feeding to the Raspberry Pi module.	Rs. 440 x 1
3. SSD 1306 Oled with SPI / I2C Support	An OLED Screen on which the information regarding the translation and face-recognition will be shown.	Rs. 200 x 1
4. FTDI Programmer	A programming module used for programming the ESP32 Camera Module and setting up the local area network.	Rs. 150 x 1
5. 30 mm Plano convex lens with a Focal Length of 100mm	For compensating the 25 cm minimum distance of vision requirement and converging rays to an appropriate distance.	Rs. 500
6. Reflecting Hand Mirror	For reflecting rays from OLED Screen to Acrylic Lens.	Rs. 100
7. 3D Printing for E.D.I.T.H frame	Provides structural support and stability.	Rs. 3000*
8. Other Accessories	Jumper Wires, Glue gun, Soldering iron etc.	Rs. 1500*
----------		Grand Total = Rs. 8,490

Detailed breakdown of the materials and components used to make SmartGaze:

Electrical/Electronics:

1. Raspberry Pi 3 : The Raspberry Pi 3 is a powerful single-board computer and is the main microcontroller that will control the operation of the SmartGaze. It will help in communication with other modules.

2. ESP32 Camera Module: The ESP32 camera module is a compact, low-cost camera module that can capture images and video. It contains an ESP32 chip with Wi-Fi and Bluetooth connectivity, an OV2640 camera sensor, and various other components.

3. OLED Screen: The OLED screen is a compact display that uses organic light-emitting diodes to produce high-contrast, high-resolution images. It is used to display the translated text generated by SmartGaze.

4. Power bank: The power bank is used to power the smart glasses, so that we can use it everywhere without interruption

Mechanical:

1. Frame: The frame is the main body of the smart glasses that will hold all the electronic components. It is designed to be lightweight and comfortable to wear.

2. Nose Bridge: The nose bridge is the part of the frame that sits on the wearer's nose. It is designed to be adjustable to fit a range of face shapes and sizes.

3. Hinges: The hinges are the joints that connect the frame to the arms of the glasses. They are designed to be durable and allow for smooth movement of the arms.

4. Lens: The lens is the part of the glasses that covers the eyes. It is designed to be scratch-resistant and provide clear vision.

Manufacturing Process Description:

We would not need manufacturing processes for the electrical components as most of them are pre-fabricated components, the frames and other mechanical parts could be manufactured with the help of 3D printing.

Assembly:

Description of the assembly process for the smart glasses:

1. First, the frame components (front frame, temple arms, and nose bridge) are assembled together. This may involve inserting screws or snap-fitting the components together.

2. The camera module, OLED screen, and Wi-Fi/Bluetooth module are then mounted onto the front frame using screws or adhesive.

3. The Raspberry Pi 3 board is mounted onto the temple arm using screws.

4. The power bank is inserted into the back of the Smartgaze and connected to the Raspberry Pi 3.

5. The hinges are attached to the temple arms and the front frame using screws.

6. The lens is inserted into the frame and secured using clips or adhesive.

7. The completed electronics module (front frame, camera module, OLED screen, Wi-Fi/Bluetooth module, and Raspberry Pi 3 board) is then connected to the power bank and tested for functionality.

8. Finally, the temple tips are attached to the temple arms to provide a comfortable fit for the user.

Limitations:

Limitations in translation:

1. The accuracy of the real-time translation may not be accurate everytime, it may have slight differences from what the actual translation is.
2. Font size will also affect the translation, if the font size is very small it would be difficult to do OCR and translate it, and the major reason for this is the camera quality we have, we are currently using a 2 megapixel camera that can perform translation on a minimum font size of 18.
3. Font type will also affect the translation, if the font type is not straight and is cursive or of some stylish font it would be difficult to do OCR.
4. Contrast of the background of the text would also affect the translation.
5. The smart glasses may not be compatible with all languages or may have difficulty recognizing complex phrases or idiomatic expressions.

Product Perspective:

1. Power consumption:

   One of the major challenges of smart glasses is their power consumption. Due to the processing power required for real-time translation, the battery may drain quickly, leading to a short operational time.

2. Weight balance:

   Since we are using microcontrollers, camera modules, we will have to place them in such a way that weight is balanced and is light in weight.

3. Camera quality:

   Camera quality will also affect the translation, currently we are using a camera of 2 megapixel and camera quality plays an important role in doing OCR and translation.

4. User Comfort:

   The weight and fit of the smart glasses could be a limitation, making them uncomfortable for long-term use.

Challenges:

In this section we will discuss the challenges faced while making our project and how we overcome these challenges.

1. Initially, we thought to use a near eye micro OLED projector, that will be used for creating augmented reality (AR) experience that can be viewed directly by the user and to overlay digital information, such as text, graphics, and video, onto the real world.

   But this technology is still in research phase, so instead of this we decided to use a simple OLED display with a camera module and a microcontroller.

2. Initially, we planned on using Raspberry pi zero W as a microcontroller because it has its own compatible camera and built-in Wi-Fi and bluetooth module, but due to its unavailability we were compelled to search for other possible microcontrollers.

3. After having a small discussion over this issue of unavailability of Raspberry pi we came to a conclusion of switching to an Arduino board which neither had an in-built wifi or bluetooth module nor a good quality camera.

4. So, in order to solve the issue of the Wifi module and camera we came up with a solution of using a ESP 32 cam module along with Arduino Mega which has an in-built 2MP camera and also an in-built wifi module.

5. Later on after successfully programming the ESP 32 cam module with the Arduino Mega, we got the video stream.

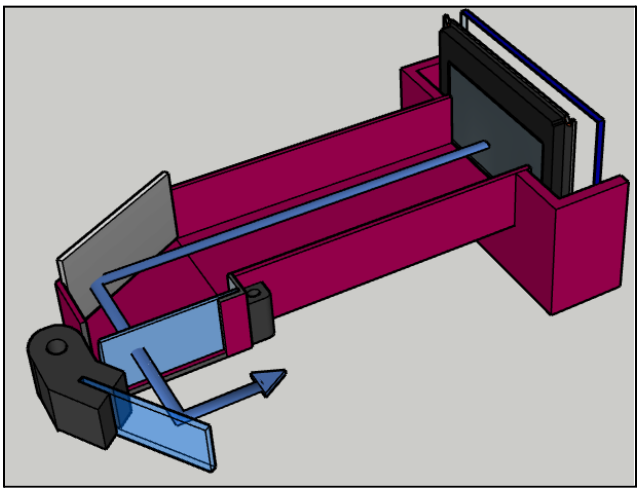
6. We tried integrating both the boards but failed to do so, and upon searching for an optimal solution we came to discover something called a FTDI programmer which has the capability to replace the Arduino board.

7. After merging all the codes, we aimed to automate their execution by running them continuously on a server. Initially, we attempted using Google Cloud for this purpose. However, we encountered an error stating that the code couldn't establish a connection due to our utilization of a local area network (LAN). Consequently, we made the decision to switch to a Raspberry Pi.

8. We successfully automated all the codes using a Raspberry Pi. When it came to choosing between two options for triggering the code execution, we opted for using a push button instead of having the code run every 5 seconds. The reason behind this decision was that if we relied on the code running at regular intervals, the person involved would not have control over when the picture was taken. This could result in blurry or unsuitable images for the subsequent code execution.

9. Another challenge we encountered was the issue of powering the Raspberry Pi. Initially, we couldn't find a suitable 5V battery for this purpose. As an alternative solution, we considered using a 12V battery with a buck converter to step down the voltage. However, after careful consideration, we ultimately decided to use a power bank to power the Raspberry Pi instead.

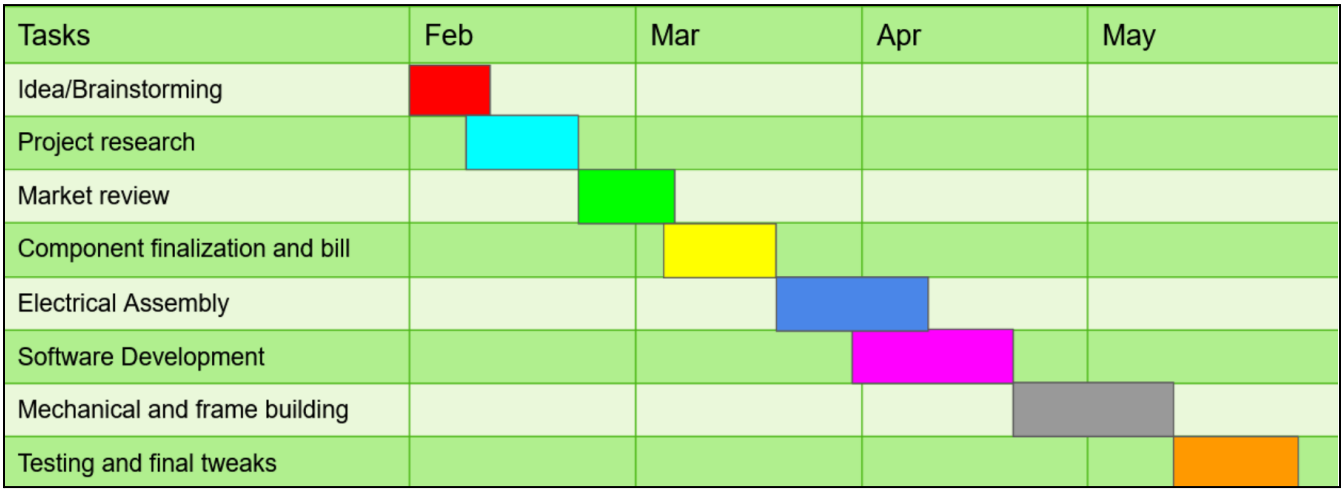
10. We encountered another challenge regarding how to display the translated text effectively. To address this, we devised a solution that involved utilizing a mirror, lens, and an OLED screen.



Scheduling Plan:

1. Idea and brainstorming - 2 weeks
2. Project research and market review- 3 weeks
3. Component finalization and bills of material - 2 weeks
4. Material sourcing and procurement - 1 weeks
5. Assembly of electronic components (Raspberry Pi 3B Module , camera module and OLED screen) - 2 weeks
6. Working on software, translation part and facial recognition- 3 weeks
7. Fabrication of mechanical components (frame, temple arms, nose bridge, and hinges) - 2 weeks
8. Final assembling of electrical, mechanical parts with the software - 2 weeks
9. Quality control and testing - 1 weeks

Following is the gantt chart of our scheduling plan.



Contribution:

BrainStorming and Deciding of Problem Statement finalization was done with the equal contribution of all of us.

1. **Electrical/Electronics:**
    a. **Connections and Circuits :** Mantavya, Saqib, Saurav, Akarshan
    b. **Raspberry Pi 3B Module:** Akarshan, Mantavya, Bhumesh, Aayushi
2. **Software :**
    a. **Tesseract Module :** Akarshan, Saqib
    b. **Translation Module :** Bhumesh, Mantavya
    c. **Face recognition :** Akarshan, Bhumesh
3. **Mechanical Aspect :**
    a. **AutoCAD :** Aayushi, Saurav
4. **Optical Aspect :** Saurav, Aayushi, Saqib

Overall, everybody had knowledge of what was going on in different fields of the projects, and everyone tried to contribute as much as they could in that particular field. This led to each member learning some new knowledge instead of just being confined to a single field.

Conclusions:

In conclusion, the development of a SmartGaze is a complex yet challenging task that requires expertise in computer science, electronic, civil and mechanical engineering.

The project has the potential to provide a convenient and efficient solution for language translation in real-time, which could have significant applications in various fields, including tourism, business, and international communication.

Despite the challenges and limitations discussed earlier, the completion of this project successfully could provide an opportunity for innovation and advancements in wearable technology, contributing to the field of augmented reality and human-computer interaction.

References

[1]

https://www.raspberrypi-spy.co.uk/2018/04/i2c-oled-display-module-with-raspberry-pi/

[2]

https://arduino-er.blogspot.com/2020/09/program-esp32-cam-using-ftdi-adapter.html

[3]

https://www.raspberrypi.com/documentation/computers/linux_kernel.html

[4]

https://chat.openai.com/ (ChatGpt)

https://github.com/tesseract-ocr/tesseract

[5]

https://bc-robotics.com/tutorials/setting-cron-job-raspberry-pi/