Virtual Smart Interaction for Physically Challenged

INDEX

ABSTRACT

The world today is largely dependent on technology and the internet, which has made it easier for people to access information, communicate with others, and perform various tasks. However, there are still many people who face challenges in accessing digital platforms, particularly those who are physically disabled, such as the deaf, blind, or dumb.

The main objective of this project is to create a user-friendly AI technology that enables physically disabled individuals to access digital platforms virtually. This will be achieved through the use of image and speech processing, which will facilitate real-time communication between disabled people using a desktop or laptop.

The proposed solution aims to bridge the communication gap between disabled individuals and the digital world, empowering them to lead more independent lives. This project will provide a comprehensive approach to solving the problems faced by the disabled and contribute towards making technology more accessible and inclusive.

Overall, this project has the potential to make a significant impact in the lives of disabled individuals and is an important step towards creating a more equitable and inclusive society.

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION IN PROJECT:

In today's world, the use of technology has become increasingly prevalent, with digital platforms being an integral part of our lives. However, for individuals who are physically challenged, such as those who are deaf, dumb or blind, accessing digital platforms can be a challenging task. To address this issue, there has been a growing interest in developing assistive technologies that can make it easier for physically challenged individuals to interact with digital platforms. One such technology is the Virtual smart interaction for disabled person.

The Virtual smart interaction for disabled person is an intelligent system that utilizes a combination of natural language processing, computer vision, machine learning, and human-computer interaction to enable natural and intuitive interaction between humans and machines. The system aims to enhance the efficiency, convenience, and personalization of human-computer interaction. With the help of this system, physically challenged individuals can access digital platforms using virtual assistants, virtual hand mouse, and virtual eye mouse.

The Virtual smart interaction for disabled person has enormous potential to revolutionize the way we interact with technology. It can enable us to communicate with our devices in a more natural and intuitive way, making it easier for us to access and utilize digital platforms. Moreover, it can provide a more personalized experience by learning from our interactions and adapting to our preferences.

1.2 PURPOSE OF THE SYSTEM:

The purpose of the Virtual smart interaction for disabled person is to create a user-friendly technology that helps physically challenged individuals access digital platforms easily. By using natural language processing, computer vision, machine learning, and human-computer interaction, the system aims to enable natural and intuitive communication between humans and machines. Its applications include virtual assistants, virtual hand mouse, virtual eye mouse and virtual key board, which enhance the efficiency, convenience, and personalization of human-computer interaction. Overall, the system aims to improve technology accessibility for everyone and revolutionize the way we interact with it.

CHAPTER 2

SYSTEM ANALYSIS

2.1 ANALYSIS MODEL:

In the analysis model of the Virtual smart interaction for disabled person, the primary objective is to identify the requirements and specifications of the system. This phase involves analyzing the problem statement, determining the system's scope, and identifying its functional and non-functional requirements.

To initiate the analysis, a comprehensive review of the existing assistive technologies available for physically challenged individuals will be conducted. This review will assist in identifying the strengths and weaknesses of these technologies and determine where the Virtual smart interaction for disabled person can add value.

The scope of the Virtual smart interaction for disabled person will be defined by identifying the target audience, specific disabilities that the system will address, and the digital platforms it will be compatible with.

After defining the scope, the functional and non-functional requirements of the system will be identified. The functional requirements will outline the system's features and capabilities, such as its ability to recognize speech, interpret visual cues, and generate responses. The non-functional requirements will specify the system's performance and usability characteristics, such as its speed, accuracy, and user interface design.

2.2 HARDWARE REQUIREMENTS:

The following hardware requirements are necessary for the Virtual smart interaction for disabled person:

- ✓ Operating System: Windows 8/8.1/10/11
- ✓ RAM: Minimum of 8 GB
- ✓ Hard Disk: Minimum of 512 GB with at least 20 GB of free space
- ✓ Processor: 12th Gen Intel(R) Core(TM) i5-1235U 1.30 GHz (or higher)
- ✓ System type: 64-bit operating system, x64-based processor.

2.3 SOFTWARE REQUIREMENTS:

The following software requirements are necessary for the development of the Virtual smart interaction for disabled person:

- ✓ Front End: Html, CSS, JavaScript
- ✓ Back End: Python and Golang

Operating System	Windows 10/11
Coding Language	Python Language
Software Tool	Visual Studio Code

2.4 PROBLEM OF THE STATEMENTS:

2.4.1 THE PROBLEMS:

The Virtual smart interaction for disabled person aims to address several problems faced by physically challenged individuals. These problems include limited access to information and communication, difficulty in performing everyday tasks, and social isolation. Additionally, traditional assistive technologies can be expensive and not easily customizable to individual needs, further exacerbating these issues. The Virtual smart interaction for disabled person seeks to address these challenges by providing an affordable, personalized, and intuitive solution that can improve the quality of life for physically challenged individuals.

2.4.2 THE SOLUTION:

The Virtual smart interaction for disabled person aims to address the following problems faced by physically challenged individuals:

- ✓ **Limited accessibility to technology:** Many assistive technologies available today are not easily accessible to physically challenged individuals. They may require additional devices or adaptations that can be expensive and difficult to obtain.
- ✓ **Limited communication options:** Physically challenged individuals may have difficulty communicating through traditional means, such as speaking or typing. This can lead to limited communication options and hinder their ability to interact with the world around them.

The solution to these problems is the development of a Virtual smart interaction for disabled person that utilizes advanced technologies such as speech recognition and natural language processing to facilitate communication and interaction with digital devices. This system will provide physically challenged individuals with an intuitive and accessible way to interact with technology, reducing the barriers to their participation in daily life activities. The system will also offer a range of customizable options to cater to the specific needs of individuals with different disabilities, ensuring that they can communicate and interact with the world around them in a more efficient and effective manner.

CHAPTER 3

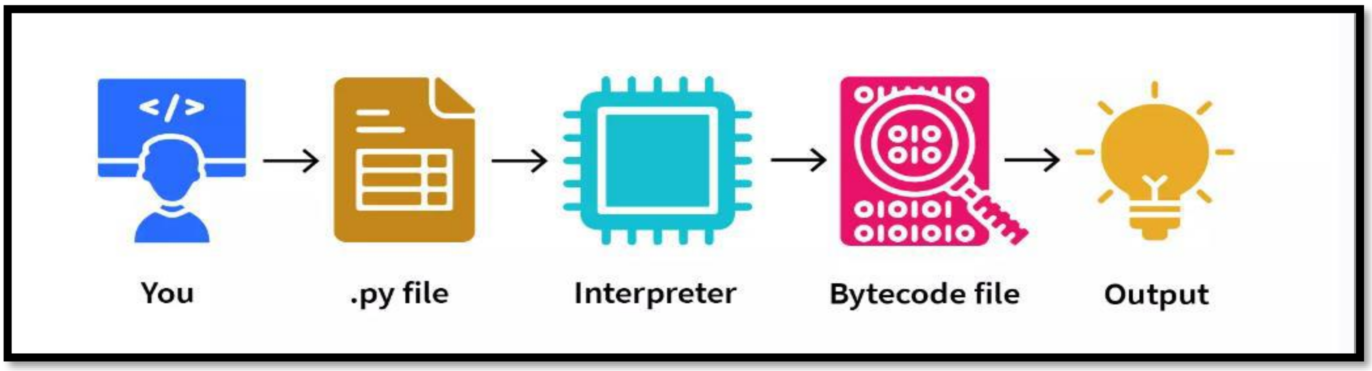
SYSTEM DEVELOPMENT ENVIRONMENT

3.1 INTRODUCTION OF PYTHON:

Python is a high-level, interpreted programming language that was first released in 1991. It is known for its simplicity, ease of use, and readability, making it an ideal language for beginners and experts alike. Python is a general-purpose language, meaning it can be used for a wide range of applications, including web development, scientific computing, data analysis, artificial intelligence, and more.

Python is dynamically typed, which means that variables do not need to be declared before use, and their type can change during runtime. It also supports object-oriented programming, as well as functional programming paradigms. The language includes a large standard library, which makes it easy to perform common tasks without having to write a lot of code.

Python's popularity has grown rapidly over the years, and it is now one of the most widely used programming languages in the world. Its success can be attributed to its simplicity, readability, and flexibility, as well as its large and supportive community of developers who contribute to its ongoing development and improvement.



The Process to generate the output:

- ✓ Write a high-level python code.
- ✓ Save the code in .py file.
- ✓ Inteprete the code.
- ✓ It will generate a bytecode file,
- ✓ The output will get printed on the screen.

3.2 GOLANG INTRODUCTION:

Go, also known as Golang, is an open-source programming language developed by Google in 2007. Go is designed to be a fast, efficient, and reliable language for writing large-scale software systems. It has a simple and concise syntax, making it easy to read and write code. Go is a statically typed language, which means that type checking is performed at compile-time, leading to faster execution and better memory management.

Go is widely used for developing network and web-based applications, as it includes features such as built-in concurrency support and a lightweight thread model. It also has excellent support for writing and reading files, working with databases, and creating and consuming APIs. Go has a rich standard library, which includes packages for encryption, compression, and other common tasks.

Go is cross-platform and can be compiled to run on different operating systems and hardware architectures. Its popularity is growing rapidly, and it is frequently used by developers who want a language that can handle large-scale applications and systems.

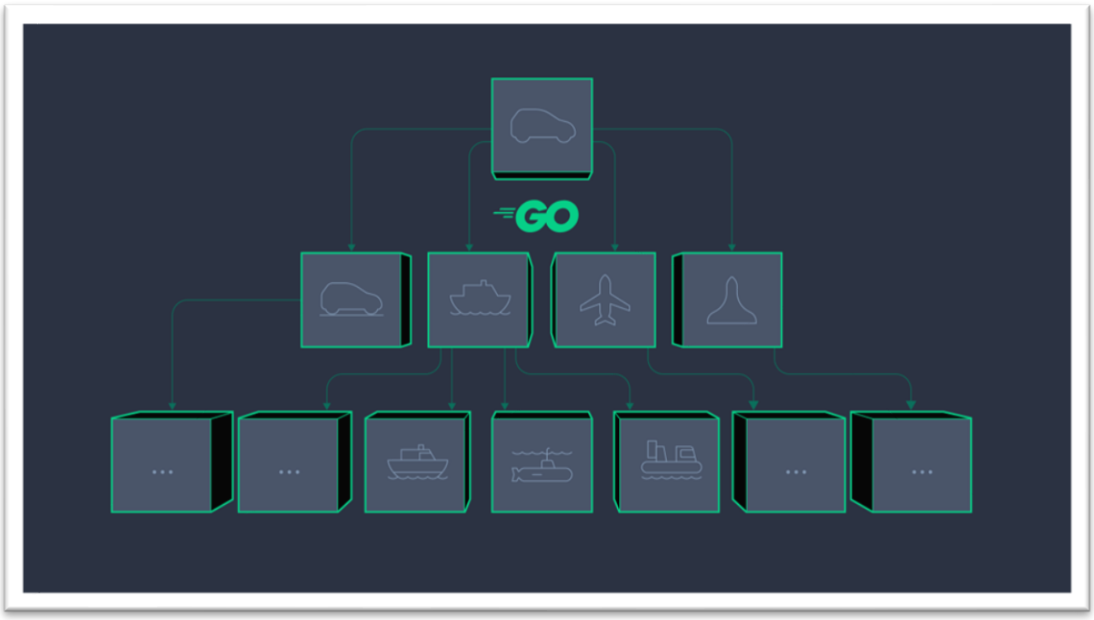


Fig:3.2.1 OOP IN GOLANG

3.3 HTML, CSS AND JAVASCRIPT INTRODUCTION:

HTML

HTML stands for Hyper Text Markup Language.Html is the standard markup language for creating Web pages. Html describes the structure of a Web page. Html consists of a series of elements. Html elements tell the browser how to display the content. Html elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

HTML is the language for describing the structure of Web pages. HTML gives authors the means to: Publish online documents with headings, text, tables, lists, photos, etc. Retrieve online information via hypertext links, at the click of a button. HTML can be used to display any type of document on the host computer, which can be geographically at a different location. It is a versatile language and can be used on any platform or desktop. There is a range of HTML tags, they help you to design your web page.

Basic Html Tags:
<!-- --> → Defines a Comment.
<a> …..</a> → Defines a hyperlink.
<body>…..</body> → Contains all tags and text in the Html-document.
<center>…..</center> → Centred text.
<div>…..</div> → Defines a section in a document.
<form>…..</form> → Defines a html form for user input.
<nav>……</nav> → Defines navigation links.
<table>…..</table> → Creates a table.
<td>…..</td> → Indicates table data in a table.
<tr>…..</tr> → Designates a table row.
<th>……</th> → Creates a heading in a table.

CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is the language for describing the presentation of Web pages, including colours, layout, and fonts. It allows one to adapt the presentation to different types of devices, such as large screens, small screens, or printers. CSS is independent of HTML and can be used with any XML-based markup language.

JAVASCRIPT

JavaScript is the Programming Language for the Web. JavaScript can update and change both HTML and CSS. JavaScript can calculate, manipulate and validate data. JavaScript is a scripting or programming language that allows you to implement complex features on web pages every time a web page does more than just sit there and display static information for you to look at displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes. JavaScript is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else.

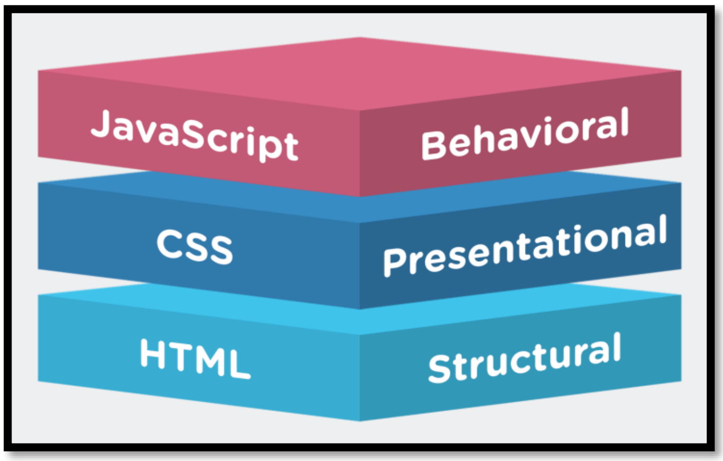


Fig:3.3.1 HTML, CSS and JavaScript

CHAPTER 4

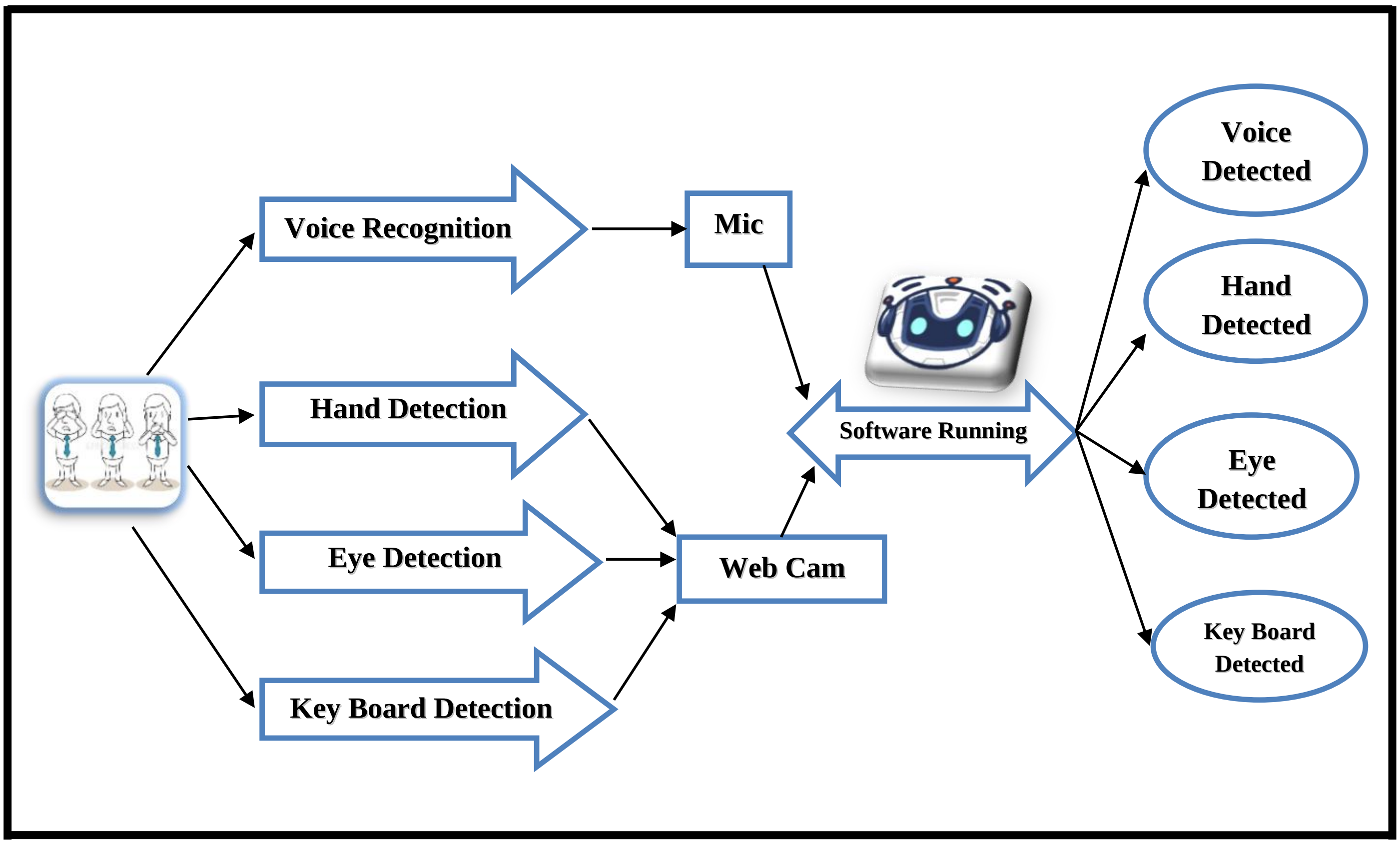
SYSTEM ARCHITECTURE

4.1 DATA FLOW CHART:

The data flow diagram for the Virtual smart interaction for disabled person is illustrated below. The system takes user input through various input channels such as voice commands, keyboard inputs, and camera interactions. The input is then processed and analyzed by the system's AI algorithms, which use natural language processing and machine learning techniques to interpret and understand the user's intentions.

The processed input is then used to generate appropriate output, which can be in the form of text, speech, or visual cues. The system can also interact with external systems and devices, such as home automation systems or assistive devices, through APIs and other integration methods.

Overall, the Virtual smart interaction for disabled person's data flow enables seamless and intuitive interactions between users and the system, providing a high level of convenience and accessibility for individuals with physical challenges.



CHAPTER 5

SCOPE AND LIMITATION

Scope:

The scope of the Virtual smart interaction for disabled person is to provide assistance and support to physically challenged individuals by utilizing artificial intelligence and natural language processing. The system will allow users to interact with digital devices and perform various tasks such as sending messages, making phone calls, and accessing the internet through voice commands and visual cues. The system will be compatible with different digital platforms and will address specific disabilities such as visual and hearing impairments.

Limitations:

The Virtual smart interaction for disabled person may have some limitations that need to be considered during its development and implementation. These limitations include:

- ✓ **Limited accuracy:** Although AI algorithms have advanced significantly, there may still be instances where the system misinterprets a command or visual cue, resulting in inaccurate responses.
- ✓ **Limited vocabulary:** The system's vocabulary may be limited to a specific set of words or phrases, making it challenging to recognize certain commands or requests.
- ✓ **Dependency on internet connectivity:** The system may require a stable internet connection to function correctly, limiting its usability in areas with poor connectivity.
- ✓ **Compatibility issues:** The system may not be compatible with certain digital devices or platforms, limiting its usage for some users.
- ✓ **Cost:** The cost of implementing the system may be a limitation for some organizations or individuals, especially if specialized hardware or software is required.

CHAPTER 6

METHODOLOGY

6.1 VIRTUAL EYE MOUSE:

- ✓ A virtual eye mouse is a type of computer input device that allows users to control the movement of their computer cursor using eye movements. This technology is particularly useful for individuals who have limited mobility or are unable to use their hands to operate a traditional mouse or keyboard.
- ✓ The virtual eye mouse works by using an eye-tracking camera, which tracks the movement of the user's eyes and translates them into movements of the computer cursor. The user can select items on the screen and perform mouse clicks by blinking their eyes or holding a gaze for a specific amount of time.
- ✓ Virtual eye mouse technology has advanced significantly in recent years and has become more accessible and affordable for individuals who require this type of assistive technology. It can be particularly useful for people with conditions such as cerebral palsy, spinal cord injuries, or muscular dystrophy.

Fig:6.1.1 Virtual Eye Mouse

6.2 VOICE ASSISTANT:

- ✓ A voice assistant is a type of artificial intelligence software that responds to voice commands and provides assistance to users. Voice assistants use natural language processing (NLP) and machine learning algorithms to understand and interpret user commands, and then perform tasks or provide information based on those commands.

- ✓ Examples of popular voice assistants include Amazon's Alexa, Apple's Siri, Google Assistant, and Microsoft's Cortana. These voice assistants can perform a wide variety of tasks, such as setting reminders, making phone calls, sending messages, playing music, providing weather forecasts, and controlling smart home devices.

- ✓ Voice assistants are becoming increasingly popular as more people embrace the convenience of hands-free technology. They are particularly useful for people with disabilities or limited mobility, as they allow for easy access to technology without the need for physical interaction. Voice assistants are also being integrated into more and more devices, such as cars, smart speakers, and wearable devices, making them a ubiquitous part of modern life.

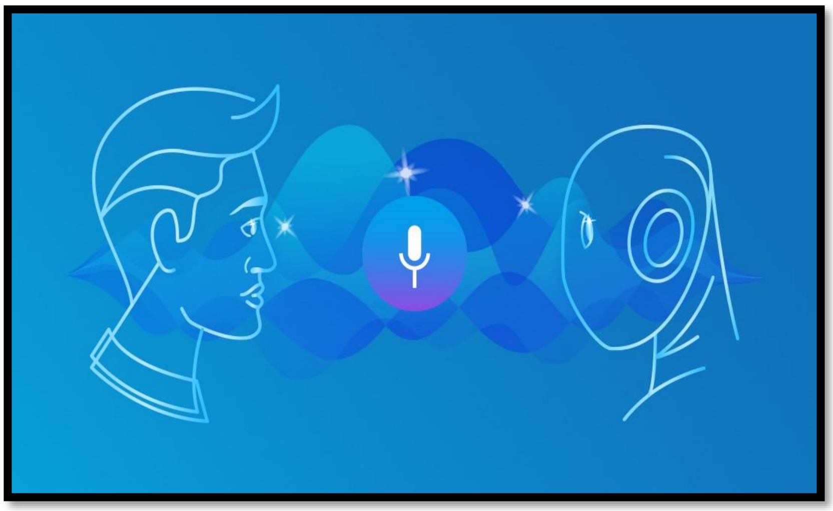


Fig:6.2.1 Voice Assistant

6.3 VIRTUAL HAND MOUSE:

- ✓ Virtual hand gestures refer to hand movements or positions that are recognized and interpreted by a computer or virtual reality system. These gestures can be used to control a virtual interface, navigate a virtual environment, or manipulate digital objects in a 3D space.

- ✓ Examples of virtual hand gestures include pointing, grabbing, rotating, swiping, or pinching. These gestures can be detected by a variety of technologies such as cameras, sensors, or motion controllers, and are often used in virtual reality systems or augmented reality applications.

- ✓ Virtual hand gestures are becoming increasingly important in a variety of fields, including gaming, education, training, and simulation, as they provide a more intuitive and immersive way of interacting with digital content.

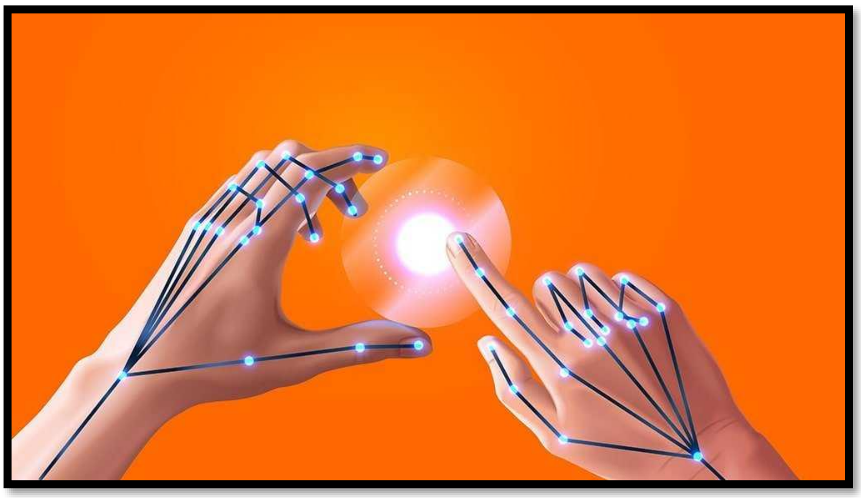


Fig:6.3.1 Virtual Hand Mouse

6.4 VIRTUAL KEYBOARD:

- ✓ Virtual Keyboard is a software component that provides a graphical interface for users to input characters without the need for a physical keyboard. It works by displaying a visual representation of a keyboard on the computer screen, allowing users to click on the keys using a mouse, touchpad, or other input device.

- ✓ The Virtual Keyboard in the Virtual smart interaction for disabled person will be designed to be accessible and user-friendly for physically challenged individuals who may have difficulty using a physical keyboard.

- ✓ It will support various input methods such as mouse clicks, touch screen, and eye tracking, and will have customizable layouts to cater to different user needs. The Virtual Keyboard will also have predictive text and autocorrect features to assist users in typing quickly and accurately.

Fig:6.4.1 Virtual Keyboard

CHAPTER 6

SYSTEM DESIGN

7.1 INPUT DESIGN:

@ VIRTUAL EYE MOUSE:

```
import cv2
import mediapipe as mp
import pyautogui

cam = cv2.VideoCapture(0)
face_mesh = mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)

screen_w, screen_h = pyautogui.size()
pyautogui.screenshot("Screenshot.png", region=(0, 0, screen_w, screen_h))

EYE_CLOSED_THRESHOLD = 3
eye_closed_counter = 0

while True:
    _, frame = cam.read()
    frame = cv2.flip(frame, 1)
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    output = face_mesh.process(rgb_frame)
    landmark_points = output.multi_face_landmarks
    frame_h, frame_w, _ = frame.shape
    if landmark_points:
        landmarks = landmark_points[0].landmark
        for id, landmark in enumerate(landmarks[474:478]):
            x = int(landmark.x * frame_w)
            y = int(landmark.y * frame_h)
            cv2.circle(frame, (x, y), 3, (0, 255, 0))
            if id == 1:
                screen_x = screen_w * landmark.x
                screen_y = screen_h * landmark.y
                pyautogui.moveTo(screen_x, screen_y)
        left = [landmarks[145], landmarks[159]]
        for landmark in left:
            x = int(landmark.x * frame_w)
            y = int(landmark.y * frame_h)
            cv2.circle(frame, (x, y), 3, (0, 255, 255))
        if (left[0].y - left[1].y) < 0.001:
            pyautogui.click()
            eye_closed_counter += 1
            if eye_closed_counter > EYE_CLOSED_THRESHOLD:
                break
    else:
        break

    cv2.waitKey(1)
```

@ VOICE ASSISTANT:

```
import ctypes
import sys
import time
import webbrowser
import googlescrap as googlescrap
import pyttsx3
import pywhatkit
import speech_recognition as sr
import datetime
import wikipedia
import os
import smtplib
import pyjokes
import subprocess
import winshell

engine = pyttsx3.init('sapi5')
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[1].id)

def speak(audio):
    engine.say(audio)
    engine.runAndWait()

def wishMe():
    hour = int(datetime.datetime.now().hour)

    if hour >= 0 and hour < 12:
        speak("Good Morning Sir !")
    elif hour >= 12 and hour < 18:
        speak("Good Afternoon Sir !")
    else:
        speak("Good Evening Sir !")
    assname = ("kera")
    speak("I am your Assistant")
    speak(assname)
```

```
def username():
    speak("How can i Help you, Sir")

def takeCommand():
    r = sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening...")
        r.adjust_for_ambient_noise(source, duration=0.2)
        recordedaudio = r.listen(source)
        r.pause_threshold = 1
        print('Done Recording..!')

    try:
        print("Recognizing...")
        query = r.recognize_google(recordedaudio, language='en-us')
        print(f"User said: {query}\n")
    except Exception as e:
        speak("Say that again please...")

        return "None"
    return query


if __name__ == "__main__":


    wishMe()
    username()
    while True:
        query = takeCommand().lower()
        if 'wikipedia' in query:
            speak('Searching Wikipedia...')
            query = query.replace("wikipedia", "")
            results = wikipedia.summary(query, sentences=2)
            speak("According to Wikipedia")
            speak(results)

        elif 'open youtube' in query:
            speak("opening youtube")
            webbrowser.open("youtube.com")

        elif 'open google' in query:
            speak("opening google")
            webbrowser.open("google.com")

        elif 'open stack overflow' in query:
            speak("opening stackoverflow")
            webbrowser.open("stackoverflow.com")

        elif 'play music' in query:
            speak("playing music")
            music_dir = 'E:\\music'
            songs = os.listdir(music_dir)
            print(songs)
            os.startfile(os.path.join(music_dir, songs[0]))

        elif 'time' in query:

            strTime = datetime.datetime.now().strftime("%H:%M:%S")

            speak(f"Sir, the time is {strTime}")

        elif 'jokes' in query:
            speak(pyjokes.get_joke())

        elif 'how are you' in query:
            speak("i am fine. Thank you")
            print("i am fine Thank you")
            speak("how are you sir")

        elif 'good' in query:
            speak("its good to know that you are fine")

        elif 'who are you' in query:
            speak("i am your personal assistant")

        elif 'google search' in query:
            query = query.replace("google search", " ")
            query = query.replace("google", " ")

            speak("This are the results of the following search")
            pywhatkit.search(query)
           try:

                pywhatkit.search(query)
                result = googlescrap.summary(query, 3)
                speak(result)
            except:
                speak(" ")

        elif 'open gmail' in query:
            speak('opening gmail')
            webbrowser.open('www.gmail.com')

        elif 'clear recycle bin' in query:
            winshell.recycle_bin().empty(confirm=False, show_progress=False, sound=True)
            speak("Recycle Bin Recycled")

        elif 'lock window' in query:
            speak("locking the device")
            ctypes.windll.user32.LockWorkStation()

        elif "hibernate" in query or "sleep" in query:
            speak("Hibernating")
            subprocess.call("shutdown / h")
        elif "log off" in query or "sign out" in query:
            speak("Make sure all the application are closed before sign-out")
            time.sleep(5)
            subprocess.call(["shutdown", "/l"])

      elif 'shutdown system' in query:
            speak("Hold On a Sec ! Your system is on its way to shut down")
            subprocess.call('shutdown / p /f')

        elif  any(word in query.lower() for word in ['quit', 'exit', 'stop', 'bye']):
            speak('okay.....!Bye Sir, have a good day.')
            sys.exit()
```

@ VIRTUAL HAND MOUSE:

```
import mediapipe as mp
import cv2
import autopy
import numpy as np
import math

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils

wCam, hCam = 320, 240
cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
wScr, hScr = autopy.screen.size()
plocX, plocY = 0, 0
clocX, clocY = 0, 0
smoothing = 6
frameR = 100
with mp_hands.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5,
max_num_hands=1) as hands:

    num_frames_without_hand = 0
    while cap.isOpened():
        success, image = cap.read()
        if not success:
            break
        image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
        image.flags.writeable = False
        results = hands.process(image)
        if results.multi_hand_landmarks:
            num_frames_without_hand = 0
        for hand_landmarks in results.multi_hand_landmarks:
            x = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].x
            y = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].y
            x1 = hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP].x
            y1 = hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP].y
            xd = x * wCam
            yd = y * hCam
            x1d = x1 * wCam
            y1d = y1 * hCam
            dist = math.hypot(x1d - xd, y1d - yd)
            h, w, c = image.shape
            x = int(x * w)
            y = int(y * h)
            cv2.rectangle(image, (frameR, frameR), (wCam - frameR, hCam - frameR), (255, 0, 255), 2)
            x = np.interp(x, (frameR, wCam - frameR), (0, wScr))
            y = np.interp(y, (frameR, hCam - frameR), (0, hScr))
            clocX = plocX + (x - plocX) / smoothing
            clocY = plocY + (y - plocY) / smoothing
            clocX = int(round(clocX))
            clocY = int(round(clocY))
            try:
                autopy.mouse.move(clocX, clocY)
                plocX, plocY = clocX, clocY
            except ValueError:
                plocX, plocY = clocX, clocY
                continue

            if dist < 45:
                autopy.mouse.toggle(down=True)
                autopy.mouse.toggle(down=False)
    else:
        num_frames_without_hand += 1
        if num_frames_without_hand > 25:
            break
cap.release()
```

@ VIRTUAL HAND MOUSE:

```
import cv2
from cvzone.HandTrackingModule import HandDetector
from time import sleep, time
from pynput.keyboard import Controller

cap = cv2.VideoCapture(0)
cap.set(3,1280)
cap.set(4,720)
detector = HandDetector(detectionCon=0.8)

keys = [["Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P"],
        ["A", "S", "D", "F", "G", "H", "J", "K", "L", ";"],
        ["Z", "X", "C", "V", "B", "N", "M", ",", ".", "/"]]

ClickedText = ""
keyboard = Controller()

def drawALL(img, buttonList, rectColor=(255, 0, 255), textColor=(255, 255, 255)):
    for button in buttonList:
        x, y = button.pos
        w, h = button.size
        cv2.rectangle(img, button.pos, (x + w, y + h), rectColor, cv2.FILLED)
        cv2.putText(img, button.text, (x + 20, y + 65), cv2.FONT_HERSHEY_SIMPLEX, 2, textColor, 3)
        return img

class Button():
    def __init__(self,pos,text,size=[80,80]):
        self.pos = pos
        self.text = text
        self.size = size

buttonList = []
for i in range(len(keys)):
    for j, key in enumerate(keys[i]):
        buttonList.append(Button([100 * j + 50, 100 * i + 50], key))

timeout = 5
start_time = time()

while True:
    success, img = cap.read()
    if not success:
        break
    img= detector.findHands(img)
    lmlist, bboxInfo = detector.findPosition(img)
    drawALL(img, buttonList, (0, 0, 0), (255, 255, 255))

    if lmlist:
        start_time = time()
        for button in buttonList:
            x,y = button.pos
            w,h = button.size
            if x<lmlist[8][0]<x+w and y<lmlist[8][1] < y+h:
                cv2.rectangle(img, button.pos, (x + w, y + h), (255, 0, 0), cv2.FILLED)
                cv2.putText(img, button.text, (x + 20, y + 65), cv2.FONT_HERSHEY_SIMPLEX, 2, (255,255, 255), 3)
                l,_,_=detector.findDistance(8,12,img)
                if l < 50:
                    keyboard.press(button.text)
                    cv2.rectangle(img, button.pos, (x + w, y + h), (0, 255, 0), cv2.FILLED)
                    cv2.putText(img, button.text, (x + 20, y + 65), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255), 3)
                    ClickedText += button.text
                    sleep(0.3)
    else:
        elapsed_time = time() - start_time
        if elapsed_time > timeout:
            print("Hand not found for more than {} seconds. Exiting program...".format(timeout))
            break
    cv2.rectangle(img, (55,345), (700,450), (255, 0, 0), cv2.FILLED)
    cv2.putText(img, ClickedText, (60,425), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255), 3)
    cv2.imshow('camera',img)
    cv2.waitKey(1)
```

@ HTML PAGE:

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8" />
    <title> AI KERA</title>
    <link rel="stylesheet" href="style.css" />
    <link rel = "icon" href = "logo.png" type = "x-icon">
    <link href="https://fonts.googleapis.com/css?family=Montserrat" rel="stylesheet">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css"   />
  </head>
  <body>
    <div class="container">
      <div id="emoji-container"></div>
      <button class="btn btn1" id="eye" onclick="Eye()">Virtual Eye Mouse</button>
      <button class="btn btn2" id="voice" onclick="Voice()">Voice Assistant</button>
      <button class="btn btn3" id="hand" onclick="Hand()">Virtual Hand Mouse</button>
      <button class="btn btn4" id="key" onclick="keyBoard()">Virtual key Board</button>
    </div>
    <script src="script.js"></script>
    <footer>
      <p>&copy; 2023 AI KERA | Designed by Elanchezhian_Muthukumar</p>
    </footer>
  </body>
</html>
```

@CSS:

```
@import url('https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;700&display=swap');
body {
  font-family: 'Open Sans', sans-serif;
  background: linear-gradient(to right, #bcd3d7, #63cff3);
  background-size: cover;
  background-position: center;
  background-attachment: fixed;
  animation: gradient-animation 10s ease infinite;
}
@keyframes gradient-animation {
  0% {
    background-position: 0% 50%;
  }
  50% {
    background-position: 100% 50%;
  }
  100% {
    background-position: 0% 50%;
  }
}
.container {
  text-align: center;
  margin-top: 290px;
  background-color: rgba(255, 255, 255, 0.8);
  padding: 30px;
}
.btn {
  border: none;
  background-clip: padding-box;
  background-image: linear-gradient(to bottom, #31c3e3, #bec2c3);
  border: 4px solid #fff;
  border-radius: 20px;
  background: linear-gradient(to right, #31c3e3, #bec2c3);
  padding: 14px 30px;
  font-size: 24px;
  font-family: "Montserrat", sans-serif;
  cursor: pointer;
  margin: 14px;
  position: relative;
  overflow: hidden;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
  transition: all 0.2s ease-out;
  color: #fff;
}
```

```
.btn:hover {
  color: #fff;
  background-color: white;
  transition: all 0.1s ease-out;
  transform: scale(1.1);
  box-shadow: 0 10px 20px rgba(0, 0, 0, 0.5);
  background-color: #ff4081;
  background-image: linear-gradient(to bottom, #ff4081, #ff9966);
}
.btn::before {
  content: "";
  position: absolute;
  left: -50%;
  width: 100%;
  height: 100%;
  background: rgba(255, 255, 255, 0.2);
  z-index: -1;
  transform: skewX(-20deg);
  transition: 0.4s;
}
.btn:hover::before {
  left: 50%;
  transition: all 0.6s cubic-bezier(0.8, -0.05, 0.2, 1.15);
}
footer {
  position: fixed;
  bottom: 0;
  left: 0;
  right: 0;
  color: #fff;
  text-align: center;
  padding: 10px;
  opacity: 0;
  animation: footerAnimation 2s ease-in-out forwards;
  background-color: rgba(0, 0, 0, 0.8);
  font-size: 18px;
}
@keyframes footerAnimation {
  100% {
    transform: scale(0.1s);
    opacity: 1;
  }
}
.animated-image {
  animation-name: zoomIn;
  animation-duration: 0.5s;
  animation-delay: 0s;
  animation-timing-function: cubic-bezier(0.175, 0.885, 0.32, 1.275);
  animation-iteration-count: 1;
}
```

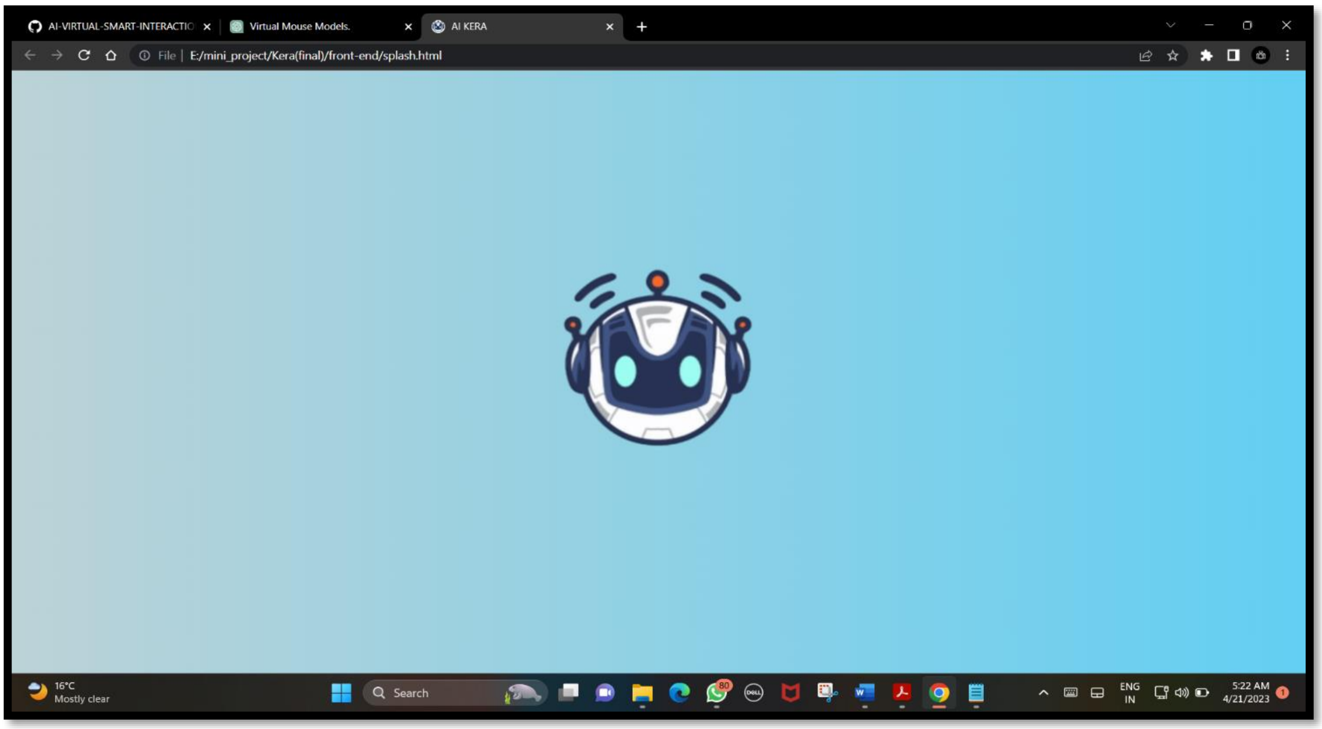
@ SCRIPT PAGE:

```
var socket = new WebSocket("ws://localhost:3333/ws");

socket.onopen = function (e) {
  socket.onmessage = function (event) {
    console.log(event);
  };
};
function Hand() {
  socket.send(JSON.stringify({ event: "hand", data: "eye access" }));
  const handImg = document.createElement('img');
  handImg.src = 'hand.png';
  handImg.alt = 'Hand Emoji';
  handImg.style.position = 'fixed';
  handImg.style.top = '50%';
  handImg.style.left = '50%';
  handImg.style.transform = 'translate(-50%, -50%)';
  handImg.classList.add('animated-image');
  document.body.appendChild(handImg);
  setTimeout(() => {
    handImg.remove();
  }, 1000);
}
function Eye() {
  socket.send(JSON.stringify({ event: "eye", data: "eye access" }));
  const eyeImg = document.createElement('img');
  eyeImg.src = 'eye.png';
  eyeImg.alt = 'Eye Emoji';
  eyeImg.style.position = 'fixed';
  eyeImg.style.top = '50%';
  eyeImg.style.left = '50%';
  eyeImg.style.transform = 'translate(-50%, -50%)';
  eyeImg.classList.add('animated-image');
  document.body.appendChild(eyeImg);
  setTimeout(() => {
    eyeImg.remove();
  }, 1000);}
```

```
function Voice() {
  socket.send(JSON.stringify({ event: "voice", data: "eye access" }));
  const voiceImg = document.createElement('img');
  voiceImg.src = 'voice.png';
  voiceImg.alt = 'Voice Emoji';
  voiceImg.style.position = 'fixed';
  voiceImg.style.top = '50%';
  voiceImg.style.left = '50%';
  voiceImg.style.transform = 'translate(-50%, -50%)';
  voiceImg.classList.add('animated-image');
  document.body.appendChild(voiceImg);
  setTimeout(() => {
    voiceImg.remove();
  }, 1000);
}

function keyBoard() {
  socket.send(JSON.stringify({ event: "key", data: "eye access" }));
  const keyboardImg = document.createElement('img');
  keyboardImg.src = 'keyboard.png';
  keyboardImg.alt = 'Keyboard Emoji';
  keyboardImg.style.position = 'fixed';
  keyboardImg.style.top = '50%';
  keyboardImg.style.left = '50%';
  keyboardImg.style.transform = 'translate(-50%, -50%)';
  keyboardImg.classList.add('animated-image');
  document.body.appendChild(keyboardImg);
  setTimeout(() => {
    keyboardImg.remove();
  }, 1000);
}
```

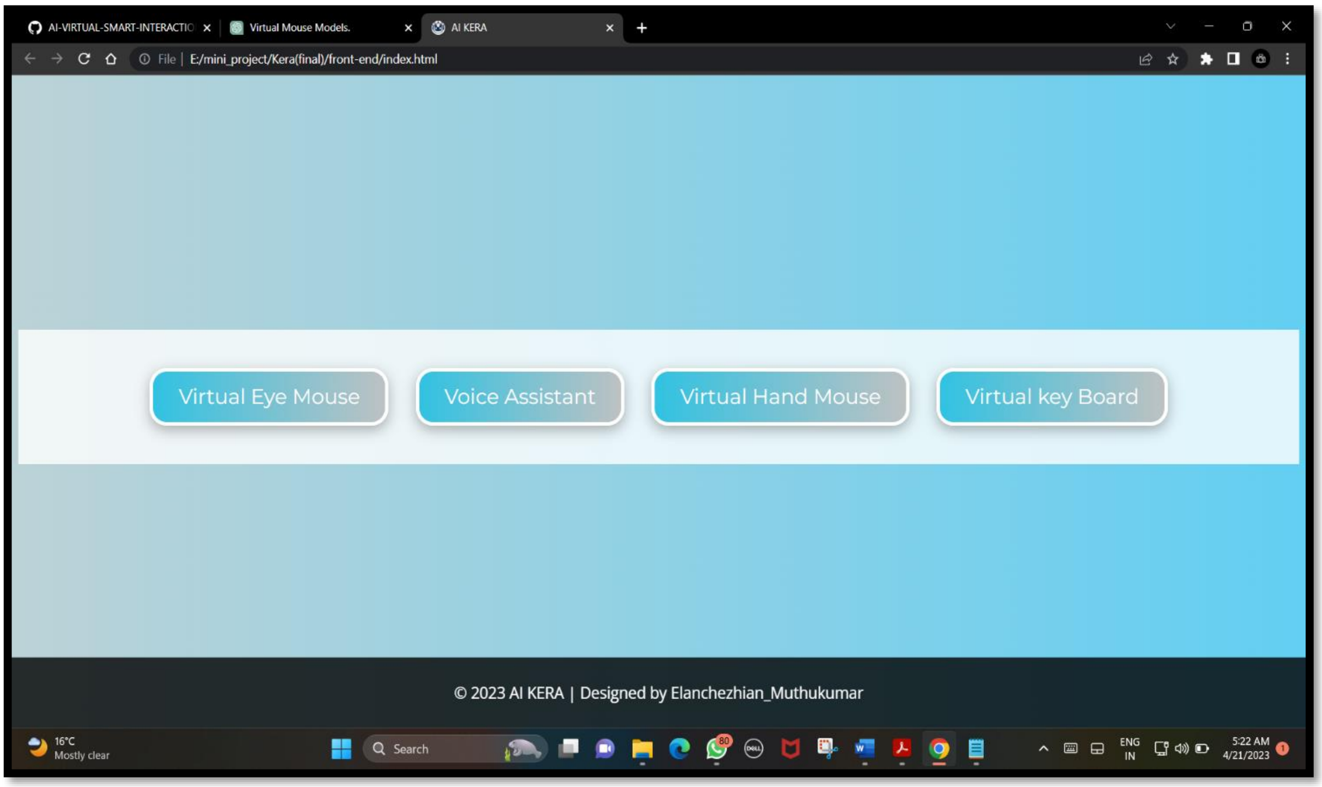
7.2 OUTPUT DESIGN:

➢ SPLASH PAGE

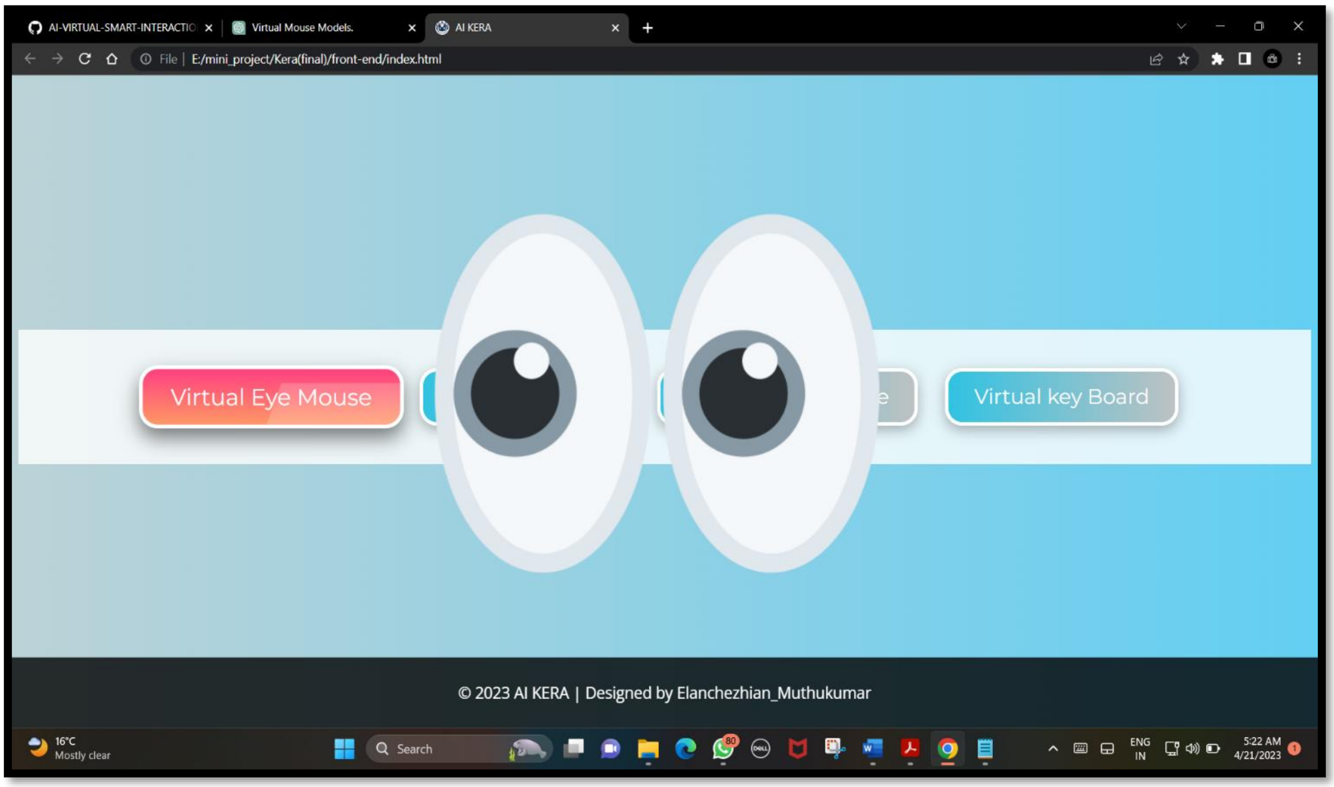


AI-VIRTUAL-SMART-INTERACTIO
Virtual Mouse Models.
AI KERA
File | E:/mini_project/Kera(final)/front-end/splash.html
16°C
Mostly clear
Search
ENG
IN
5:22 AM
4/21/2023

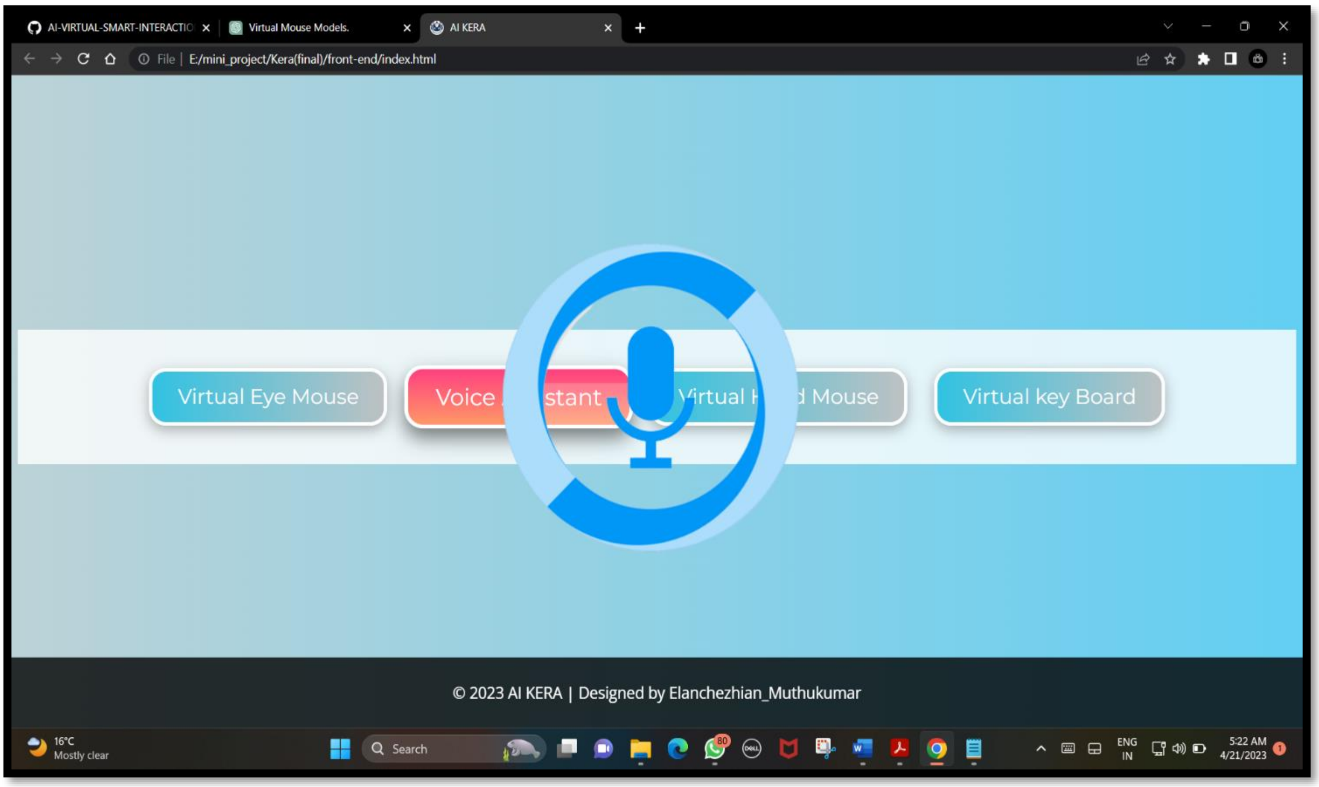
➢ HOME PAGE



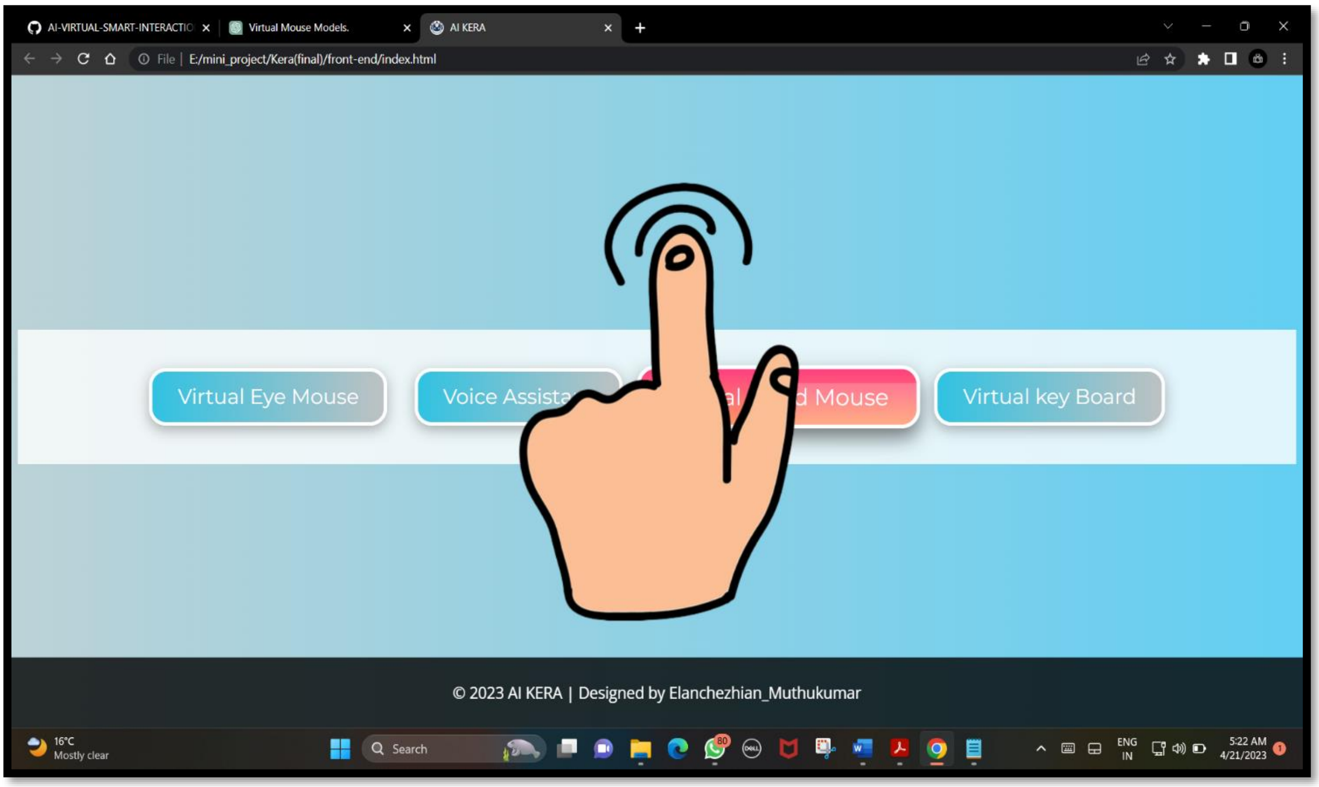
➢ VIRTUAL EYE MOUSE


AI-VIRTUAL-SMART-INTERACTIO
Virtual Mouse Models.
AI KERA
File | E:/mini_project/Kera(final)/front-end/index.html
Virtual Eye Mouse
Virtual key Board
© 2023 AI KERA | Designed by Elanchezhian_Muthukumar
16°C
Mostly clear
Search
5:22 AM
4/21/2023

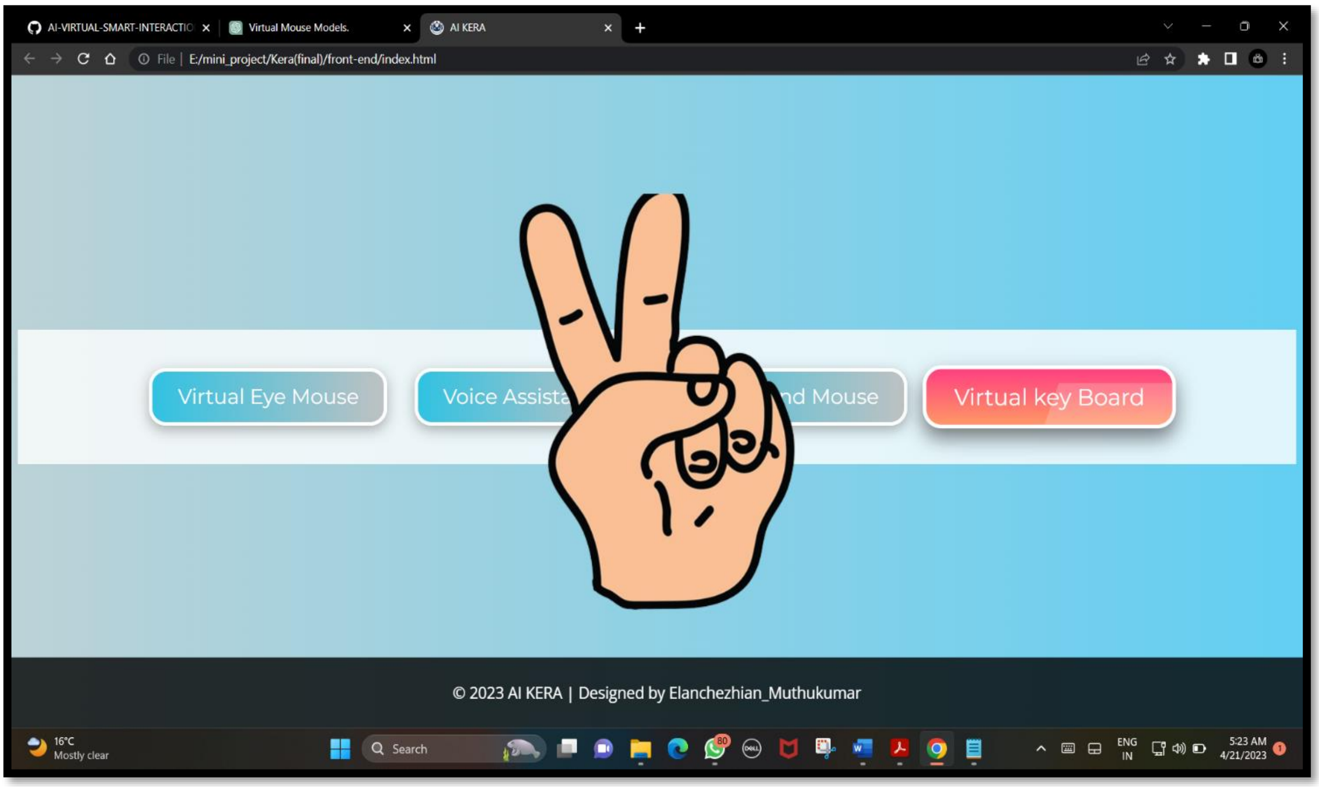
➢ VOICE ASSISTANT


AI-VIRTUAL-SMART-INTERACTIO
Virtual Mouse Models.
AI KERA
File | E:/mini_project/Kera(final)/front-end/index.html
Virtual Eye Mouse
Voice stant
Virtual H d Mouse
Virtual key Board
© 2023 AI KERA | Designed by Elanchezhian_Muthukumar
16°C
Mostly clear
Search
5:22 AM
4/21/2023

➢ VIRTUAL HAND MOUSE


AI-VIRTUAL-SMART-INTERACTIO
Virtual Mouse Models.
AI KERA
File | E:/mini_project/Kera(final)/front-end/index.html
Virtual Eye Mouse
Voice Assista
d Mouse
Virtual key Board
© 2023 AI KERA | Designed by Elanchezhian_Muthukumar
16°C
Mostly clear
Search
ENG
IN
5:22 AM
4/21/2023

➢ VIRTUAL KEY BOARD


AI-VIRTUAL-SMART-INTERACTIO
Virtual Mouse Models.
AI KERA
File | E:/mini_project/Kera(final)/front-end/index.html
Virtual Eye Mouse
Voice Assista
nd Mouse
Virtual key Board
© 2023 AI KERA | Designed by Elanchezhian_Muthukumar
16°C
Mostly clear
Search
ENG
IN
5:23 AM
4/21/2023

CHAPTER 8

SYSTEM TESTING AND IMPLEMENTATION

The implementation and testing of a virtual smart interaction system for deaf, dumb and blind people is a challenging task. The system needs to be designed in such a way that it is user-friendly and accessible to people with disabilities. The following steps can be taken for the successful implementation and testing of the system:

- ✓ **Requirements gathering:** The first step is to gather the requirements of the system. This involves understanding the needs of deaf, dumb, and blind people and identifying the features that are essential for the system.
- ✓ **Design:** Once the requirements are gathered, the next step is to design the system. The system should be designed in such a way that it is accessible and user-friendly for people with disabilities.
- ✓ **Implementation:** After the design is complete, the system can be implemented using appropriate programming languages and tools.
- ✓ **Testing:** Once the system is implemented, it needs to be thoroughly tested to ensure that it meets the requirements and is working as expected. Testing can include functional testing, performance testing, and usability testing.
- ✓ **User feedback**: After testing, the system can be presented to users for feedback. This feedback can be used to further improve the system and make it more user-friendly.
- ✓ **Deployment:** Once the system is fully tested and refined, it can be deployed for use by the target audience.
- ✓ **Maintenance:** Finally, the system needs to be maintained and updated to ensure that it continues to meet the needs of the users.

CHAPTER 9

CONCLUSION & BIBLIOGRAPHY

CONCLUSION:

Moreover, the development and implementation of the Virtual smart interaction for disabled person require a multidisciplinary approach involving experts in AI, computer science, engineering, and user experience design. Collaboration and cooperation among these experts are crucial for the success and effectiveness of the system. In addition, ethical considerations and data privacy should be taken into account during the development and deployment of the system.

Overall, the Virtual smart interaction for disabled person has the potential to make a significant positive impact on the lives of people with disabilities, and it represents a crucial step towards creating a more inclusive and accessible society. As researchers and developers continue to improve and expand the capabilities of this technology, we can expect it to become a valuable tool for promoting equity, diversity, and inclusion in various industries and fields.

BIBLIOGRAPHY:

Website

- ✓ https://docs.python.org/3/tutorial/
- ✓ https://www.geeksforgeeks.org/opencv-python-tutorial/
- ✓ https://go.dev/
- ✓ https://www.w3schools.com/html/html_scripts.asp
- ✓ https://github.com/Elanchezhian2712/VIRTUAL-SMART-INTERACTION-FOR-DISABLED-PEOPLE