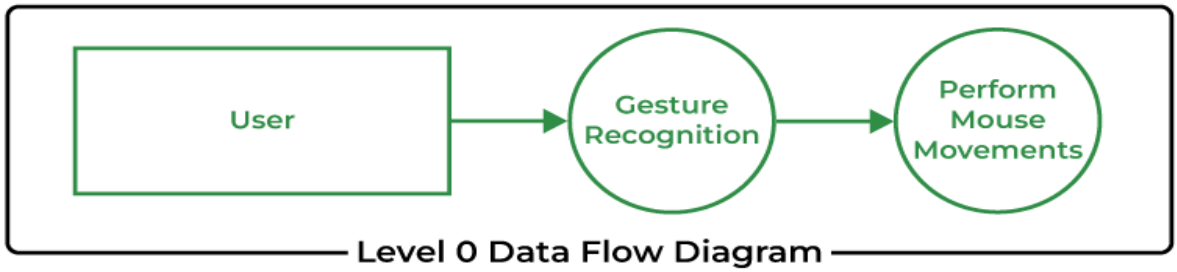
TABLE OF CONTENT

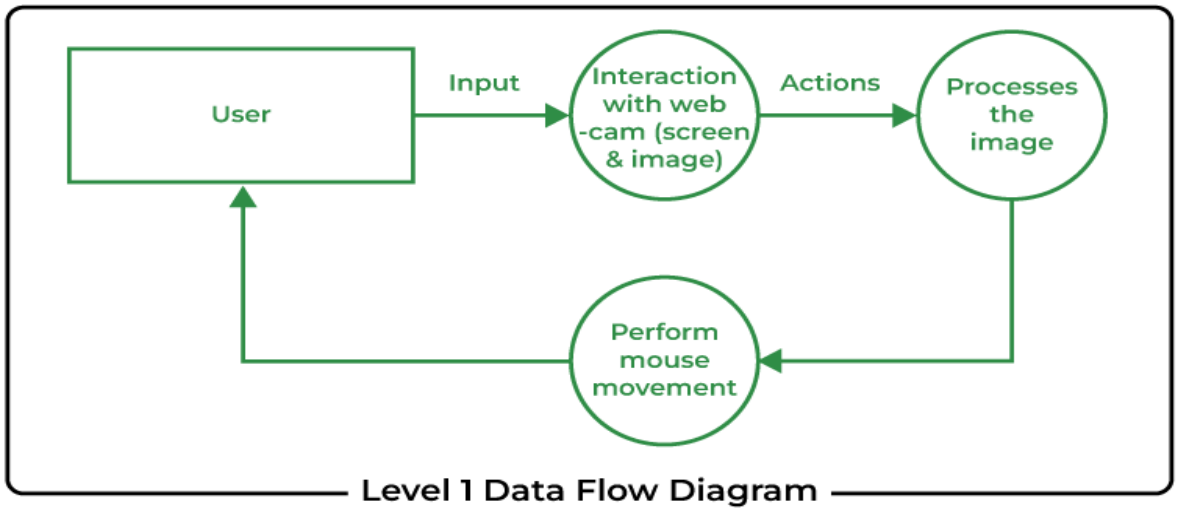
Virtual Mouse Control Using Eye Tracking and Voice Recognition

INTRODUCTION:

This document provides a comprehensive overview of a virtual mouse control system developed using Python. The system leverages eye tracking technology to pinpoint the user's gaze on the screen, enabling mouse movement control. Additionally, voice recognition capabilities allow users to execute commands and actions through vocal input.

Architecture:





SYSTEM SPECIFICATION:

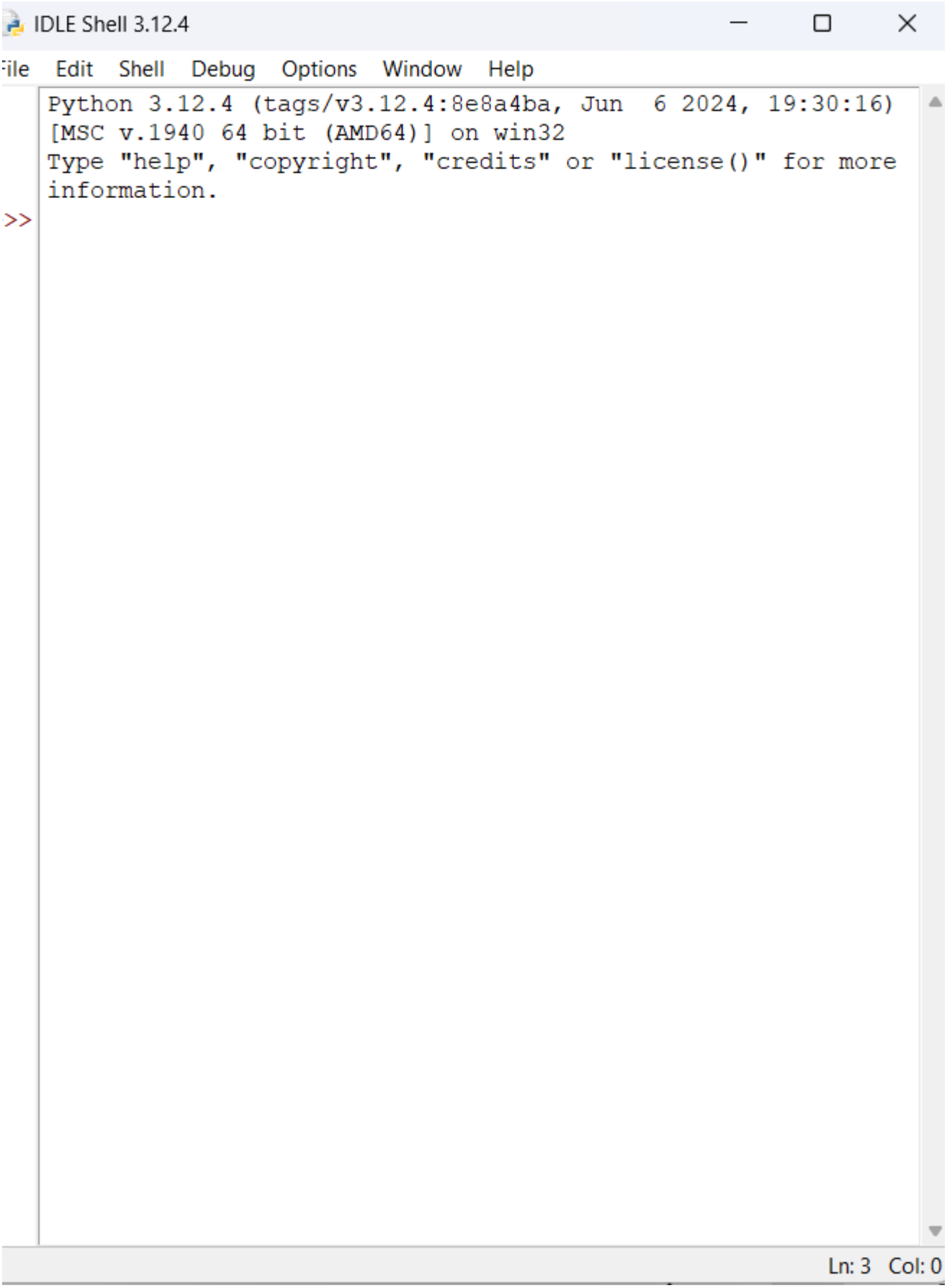
The system specification outlines the hardware and software requirements necessary for the successful operation of the virtual mouse control system. This includes a detailed breakdown of the hardware components and the specific software libraries utilized for the system's functionality

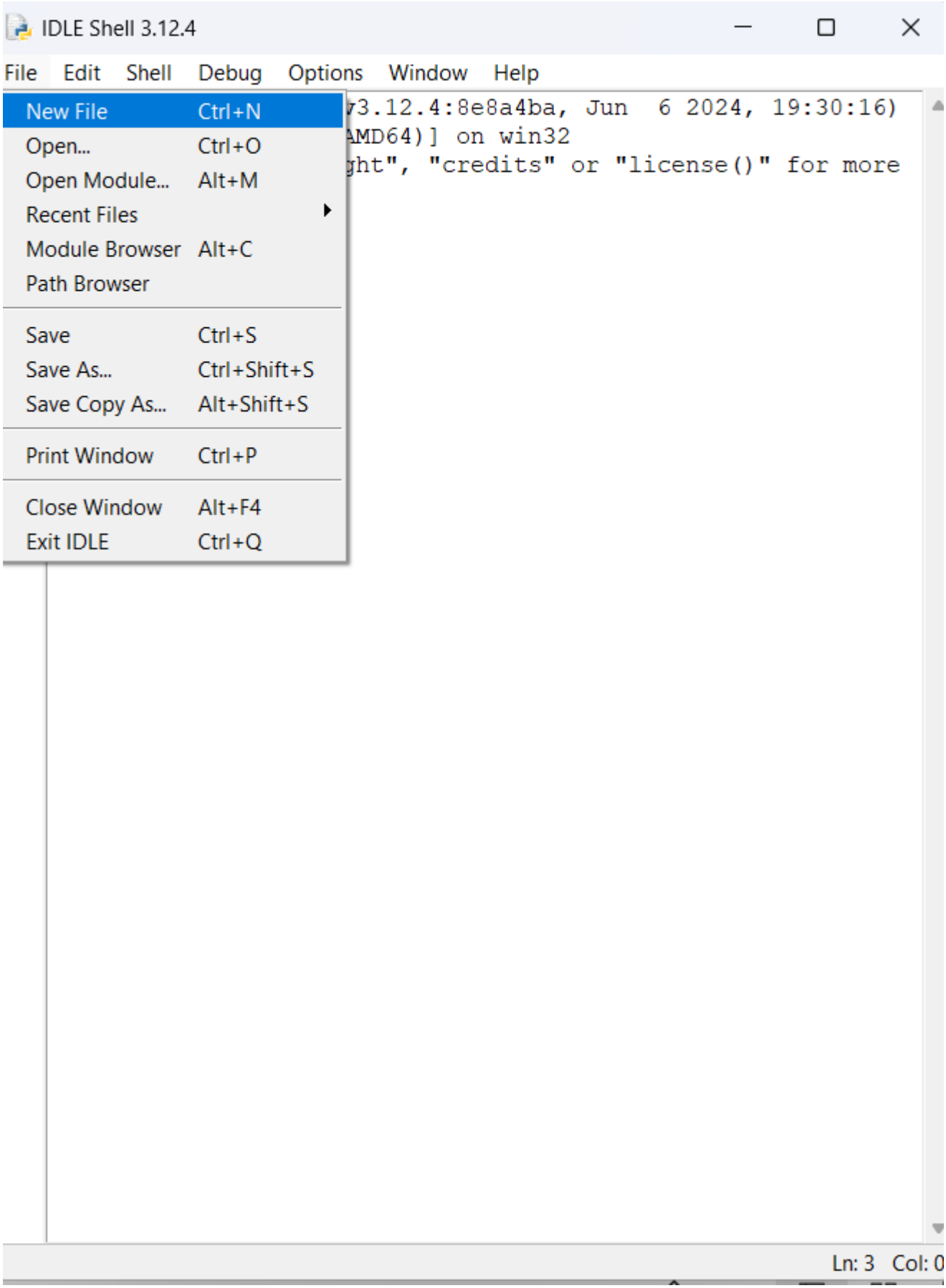
HARDWARE CONFIGURATION

The hardware configuration for the virtual mouse control system comprises several key components that work together to enable seamless operation. These components include a computer with sufficient processing power to run the system, an eye tracking device capable of accurately detecting and tracking the user's gaze, a microphone for voice recognition input, and a display screen.

SOFTWARE SPECIFICATION:

The software specification delves into the specific software libraries and frameworks employed in the development of the virtual mouse control system. These libraries provide the foundation for the system's functionalities, including eye tracking, voice recognition, and system communication. The software specification will outline the specific libraries utilized for each aspect of the system.


IDLE Shell 3.12.4
File Edit Shell Debug Options Window Help
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun 6 2024, 19:30:16)
[MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>
Ln: 3 Col: 0


IDLE Shell 3.12.4
File Edit Shell Debug Options Window Help
New File Ctrl+N
Open... Ctrl+O
Open Module... Alt+M
Recent Files
Module Browser Alt+C
Path Browser
Save Ctrl+S
Save As... Ctrl+Shift+S
Save Copy As... Alt+Shift+S
Print Window Ctrl+P
Close Window Alt+F4
Exit IDLE Ctrl+Q
v3.12.4:8e8a4ba, Jun 6 2024, 19:30:16)
AMD64)] on win32
ght", "credits" or "license()" for more
Ln: 3 Col: 0

SOFTWARE DESCRIPTION:

The software description provides a detailed overview of the core functionality of the virtual mouse control system. This involves outlining the key modules and their respective roles in the system's operation. The description will cover the system's user interface, the algorithm for interpreting eye tracking data, the voice recognition engine, and the integration of these components into a cohesive system.

SYSTEM STUDY:

The system study delves into the existing system and its limitations, highlighting the need for a more accessible and user-friendly alternative. The study will analyze the shortcomings of traditional mouse control methods and provide a justification for the development of the virtual mouse control system.

EXISTING SYSTEM:

The existing system, utilizing a physical mouse, presents several challenges for individuals with limited mobility. The reliance on precise hand movements and dexterity can be a significant barrier for those with physical limitations. The existing system may not be suitable for all users, prompting the development of a more inclusive solution.

EXISTING SYSTEM:

Traditional mouse control relies on physical interaction with a device, often requiring precise hand movements and coordination. This method can be challenging for individuals with disabilities, limiting their ability to interact with computers effectively. The need for a more accessible and intuitive control method led to the development of the virtual mouse control system.

DRAWBACKS:

The drawbacks of the existing system include the need for physical interaction with a mouse, which can be challenging for individuals with limited mobility. Additionally, traditional mouse control often lacks flexibility and adaptability for users with diverse needs. These limitations emphasize the need for an alternative solution that prioritizes accessibility and user-friendliness.

Eye Control Drawbacks

1. Accuracy Issues: Eye-tracking technology can be sensitive to lighting conditions, eye movements, and calibration issues, leading to inaccurate cursor movements.
2. Fatigue: Continuously using eye movements to control the cursor can cause eye strain and fatigue.
3. Limited Freedom: Users may feel constrained by the need to maintain eye contact with the screen to control the cursor.

Voice Control Drawbacks:

1. Speech Recognition Errors: Voice recognition technology can struggle with accents, background noise, and homophones, leading to incorrect commands.
2. Limited Vocabulary: Voice-controlled systems may not understand complex commands or specialized vocabulary.
3. Background Noise Interference: Background noise can interfere with voice commands, causing errors or misinterpretations.
4. Dependence on Audio Quality: Voice control relies on high-quality audio input, which can be affected by microphone quality, distance, and orientation.

Visual Mouse:

Visual Mouse is a revolutionary system that utilizes eye-tracking and voice recognition for computer control.

This system aims to enhance accessibility and offer a natural, intuitive way for users to interact with their devices.

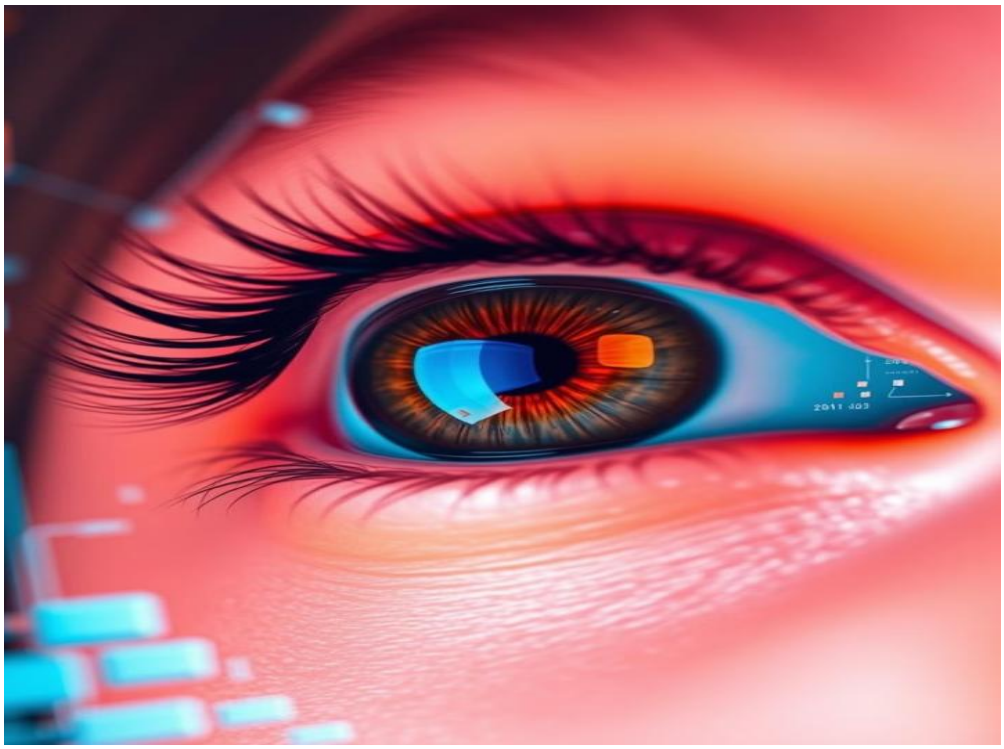
Key Features

Eye Tracking

Precision eye tracking allows for accurate cursor control without physical input.

Voice Recognition

Users can control actions and navigate menus with voice commands.



Customizable Interface

The system can be tailored to individual preferences and accessibility needs.

Compatibility

Designed to work seamlessly with various operating systems and applications.

System Architecture and Development:

Hardware:

High-resolution eye-tracking camera, microphone, and processing unit.

Software

Custom software integrates eye tracking, voice recognition, and computer control modules.

Algorithm Design

Advanced algorithms process eye movements and voice commands to translate them into computer actions.

User Interface

Intuitive interface designed for seamless interaction with the Visual Mouse system.

File Design:

Configuration Files:

Store user settings, calibration data, and system parameters.

Eye Tracking Data:

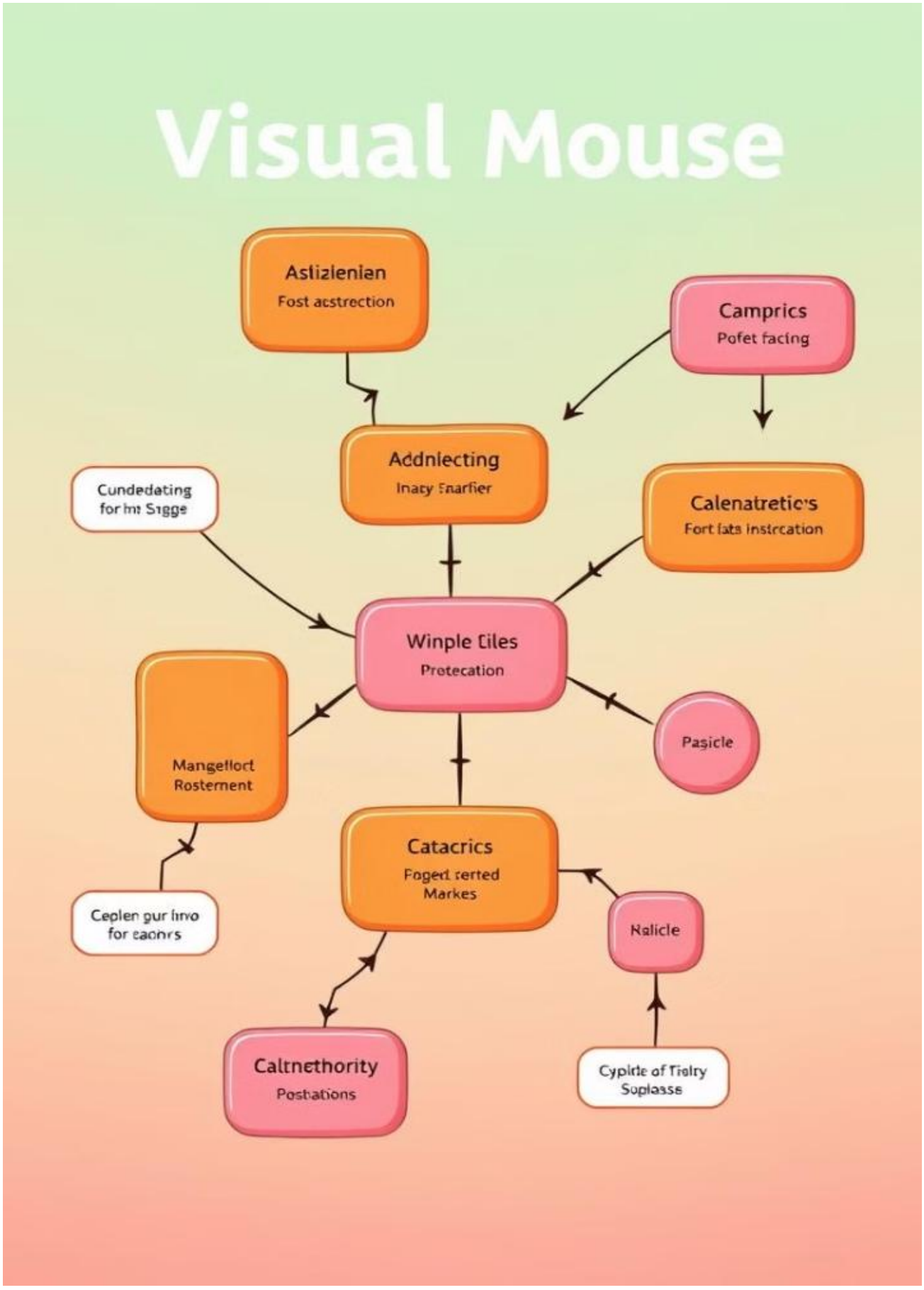
Raw eye movement data captured by the camera, processed and analyzed.

Voice Command Data:

Recorded audio data processed and converted into text commands.

Output Data:

Processed data translated into computer actions, such as cursor movement, clicks, and keystrokes.


Visual Mouse
Camprics
Addniecting
Calenatretic's
Winple Ciles
Protecation
Pasicle
Catacrics
Caltnethority
Postations
Cyplde of Tialry
Suplasse

System Modules:

Eye Tracking Module

Captures, processes, and analyzes eye movement data.

Voice Recognition Module

Processes audio input, converts speech to text, and interprets commands.

User Interface Module

Provides a user-friendly interface for interacting with the system.
Connects the Visual Mouse system to the computer's operating system and applications.

Input Design:

Eye Tracking Input:

Real-time eye movement data is captured by the camera and processed by the system.

Voice Recognition Input

Voice commands are captured by the microphone, analyzed, and translated into computer instructions.

Optional Keyboard Input

Users can still use a keyboard for text input or other actions.

Code Design:

Eye Tracking Library

openCV
mediapipe
pyautogui

Voice Recognition Library

speech_recognition
threading

Voice Recognition Engine

The core code for processing and understanding voice commands, including noise reduction and speech-to-text translation.

System Integration Module

Code that seamlessly integrates eye tracking and voice recognition with the computer's operating system and applications.

Module’s

```
import cv2
import mediapipe as mp
import pyautogui
import speech_recognition as sr
import threading
```

CODE:

```
import cv2
import mediapipe as mp
import pyautogui
import speech_recognition as sr
import threading

# Initialize the webcam and MediaPipe Face Mesh
cam = cv2.VideoCapture(0)
face_mesh =
mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)

# Get screen dimensions
screen_w, screen_h = pyautogui.size()

# Initialize recognizer
recognizer = sr.Recognizer()

# Function to listen for voice commands
def listen_for_commands():
    with sr.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source)
        while True:
            print("Listening for command...")
            try:
                # Listen to the microphone
                audio = recognizer.listen(source)
```

```
 command = recognizer.recognize_google(audio).lower()
            print(f"Voice command received: {command}")

            # Handle voice commands
            if "click" in command:
                pyautogui.click()
            elif "scroll up" in command:
                pyautogui.scroll(10)
            elif "scroll down" in command:
                pyautogui.scroll(-10)
            elif "right click" in command:
                pyautogui.rightClick()
            elif "move" in command:
                print("Say direction: up, down, left, or right")
                audio = recognizer.listen(source)
                direction =
recognizer.recognize_google(audio).lower()
                if "up" in direction:
                    pyautogui.move(0, -10)
                elif "down" in direction:
                    pyautogui.move(0, 10)
                elif "left" in direction:
                    pyautogui.move(-10, 0)
                elif "right" in direction:
                    pyautogui.move(10, 0)

    except sr.UnknownValueError:
            print("Sorry, I did not understand that.")
        except sr.RequestError as e:
            print(f"Could not request results; {e}")
```

```
# Start a new thread to listen for commands in parallel
voice_thread = threading.Thread(target=listen_for_commands)
voice_thread.daemon = True
voice_thread.start()

# Main webcam loop
while True:
    # Capture the frame from the webcam
    ret, frame = cam.read()
    if not ret:
        print("Failed to grab frame. Exiting...")
        break

    # Flip the frame horizontally for a mirror effect
    frame = cv2.flip(frame, 1)

    # Convert the frame to RGB for MediaPipe processing
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    output = face_mesh.process(rgb_frame)

    # Get the frame dimensions
    frame_h, frame_w, _ = frame.shape

    # Process landmarks if available
    if output.multi_face_landmarks:
        landmarks = output.multi_face_landmarks[0].landmark
```

```
 # Map specific landmarks for controlling the mouse
        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), -1)  # Draw
landmarks on the frame
            if id == 1:  # Use the second landmark for cursor movement
                screen_x = screen_w / frame_w * x
                screen_y = screen_h / frame_h * y
                pyautogui.moveTo(screen_x, screen_y)

        # Check for blink detection (landmarks 145 and 159)
        left_eye = [landmarks[145], landmarks[159]]
        for point in left_eye:
            x = int(point.x * frame_w)
            y = int(point.y * frame_h)
            cv2.circle(frame, (x, y), 3, (0, 255, 255), -1)  # Draw left eye
points

        # Click when the eye is almost closed
        if (left_eye[0].y - left_eye[1].y) < 0.004:
            pyautogui.click()
            pyautogui.sleep(1)
```

```
 # Display the output frame
    cv2.imshow('Eye Controlled Mouse', frame)

    # Break the loop if 'Esc' is pressed
    if cv2.waitKey(1) & 0xFF == 27:
        break

# Release resources
cam.release()
cv2.destroyAllWindows()
```

vmuv.py - C:\Users\pugaz\OneDrive\Desktop\vmuv.py (3.12.4)

File Edit Format Run Options Window Help

```
import cv2
import mediapipe as mp
import pyautogui
import speech_recognition as sr
import threading

# Initialize the webcam and MediaPipe Face Mesh
cam = cv2.VideoCapture(0)
face_mesh = mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)

# Get screen dimensions
screen_w, screen_h = pyautogui.size()

# Initialize recognizer
recognizer = sr.Recognizer()

# Function to listen for voice commands
def listen_for_commands():
    with sr.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source)
        while True:
            print("Listening for command...")
            try:
                # Listen to the microphone
                audio = recognizer.listen(source)
                command = recognizer.recognize_google(audio).lower()
                print(f"Voice command received: {command}")

                # Handle voice commands
                if "click" in command:
                    pyautogui.click()
                elif "scroll up" in command:
                    pyautogui.scroll(10)
                elif "scroll down" in command:
                    pyautogui.scroll(-10)
                elif "right click" in command:
                    pyautogui.rightClick()
                elif "move" in command:
                    print("Say direction: up, down, left, or right")
                    audio = recognizer.listen(source)
                    direction = recognizer.recognize_google(audio).lower()
                    if "up" in direction:
                        pyautogui.move(0, -10)
                    elif "down" in direction:
                        pyautogui.move(0, 10)
                    elif "left" in direction:
                        pyautogui.move(-10, 0)
```

vmuv.py - C:\Users\pugaz\OneDrive\Desktop\vmuv.py (3.12.4)

File Edit Format Run Options Window Help

```
            except sr.UnknownValueError:
                print("Sorry, I did not understand that.")
            except sr.RequestError as e:
                print(f"Could not request results; {e}")

# Start a new thread to listen for commands in parallel
voice_thread = threading.Thread(target=listen_for_commands)
voice_thread.daemon = True
voice_thread.start()

# Main webcam loop
while True:
    # Capture the frame from the webcam
    ret, frame = cam.read()
    if not ret:
        print("Failed to grab frame. Exiting...")
        break

    # Flip the frame horizontally for a mirror effect
    frame = cv2.flip(frame, 1)

    # Convert the frame to RGB for MediaPipe processing
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    output = face_mesh.process(rgb_frame)

    # Get the frame dimensions
    frame_h, frame_w, _ = frame.shape

    # Process landmarks if available
    if output.multi_face_landmarks:
        landmarks = output.multi_face_landmarks[0].landmark

        # Map specific landmarks for controlling the mouse
        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), -1)  # Draw landmarks on the frame
            if id == 1:  # Use the second landmark for cursor movement
                screen_x = screen_w / frame_w * x
                screen_y = screen_h / frame_h * y
                pyautogui.moveTo(screen_x, screen_y)

        # Check for blink detection (landmarks 145 and 159)
        left_eye = [landmarks[145], landmarks[159]]
        for point in left_eye:
            x = int(point.x * frame_w)
            y = int(point.y * frame_h)
```

vmuv.py - C:\Users\pugaz\OneDrive\Desktop\vmuv.py (3.12.4)

File Edit Format Run Options Window Help

```
    # Flip the frame horizontally for a mirror effect
    frame = cv2.flip(frame, 1)

    # Convert the frame to RGB for MediaPipe processing
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    output = face_mesh.process(rgb_frame)

    # Get the frame dimensions
    frame_h, frame_w, _ = frame.shape

    # Process landmarks if available
    if output.multi_face_landmarks:
        landmarks = output.multi_face_landmarks[0].landmark

        # Map specific landmarks for controlling the mouse
        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), -1)
            if id == 1:
                screen_x = screen_w / frame_w * x
                screen_y = screen_h / frame_h * y
                pyautogui.moveTo(screen_x, screen_y)

        # Check for blink detection (landmarks 145 and 159)
        left_eye = [landmarks[145], landmarks[159]]
        for point in left_eye:
            x = int(point.x * frame_w)
            y = int(point.y * frame_h)
            cv2.circle(frame, (x, y), 3, (0, 255, 255), -1)

        # Click when the eye is almost closed
        if (left_eye[0].y - left_eye[1].y) < 0.004:
            pyautogui.click()
            pyautogui.sleep(1)

    # Display the output frame
    cv2.imshow('Eye Controlled Mouse', frame)

    # Break the loop if 'Esc' is pressed
    if cv2.waitKey(1) & 0xFF == 27:
        break

# Release resources
cam.release()
cv2.destroyAllWindows()
```

Output Design:

Cursor Control

The system translates eye movements into precise cursor control on the screen.

Click Actions

Eye-based gestures or voice commands can trigger mouse clicks, such as left-click, right-click, or double-click.

Scrolling and Navigation

Users can scroll through documents, browse web pages, or navigate menus using eye movements or voice commands.

Output: