



PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

Project Report (Phase-1) ***on***

Cursor Pointer Control Using Eye and Hand Gestures

Submitted by
Bharath S - (PES1PG22CA044)

Oct 2023 – Jan 2024

under the guidance of

Guide Details

Mr. Santosh S Katti
Assistant Professor
Department of Computer Applications,
PESU, Bengaluru – 560085



**FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER APPLICATIONS
PROGRAM – MASTER OF COMPUTER APPLICATIONS
CERTIFICATE**

This is to certify that the project entitled

Cursor Pointer Control Using Eye and Hand Gestures

is a bonafide work carried out by

Bharath S-PES1PG22CA044

in partial fulfillment for the completion of Capstone Project, Phase-1 work in the Program of Study MCA under rules and regulations of PES University, Bengaluru during the period Oct. 2023 – Jan 2024. The project report has been approved as it satisfies the academic requirements of 3rd semester MCA.

Signature with date

Internal Guide

Mr. Santosh S Katti

Assistant Professor

Department of Computer Applications,
PES University, Bengaluru - 560085

Signature with date & Seal

Chairperson

Dr. Veena S

Department of Computer Applications,
PES University, Bengaluru - 560085

DECLARATION

I, **Bharath S**, bearing **PES1PG22CA044** hereby declare that the Capstone project phase-1 entitled, ***Cursor Pointer Control Using Eye and Hand Gestures***, is an original work done by me under the guidance of **Mr. Santosh S Katti**, Assistant Professor, PES University, and is being submitted in partial fulfillment of the requirements for completion of 3rd Semester course in the Program of Study **MCA**. All corrections/suggestions indicated for internal assessment have been incorporated in the report.

PLACE:Banglore

DATE: 29-01-2024

Bharath S

Contents

1.Introduction	1
1.1 Problem Scenario	1
1.2 Proposed Solution.....	2
1.3 Scope	2
2.Literature Survey	3
2.1 Domain Survey	3
2.2 Existing Systems	4
2.3 Tools and Technologies	6
3.Hardware and Software Requirements.....	8
3.1 Hardware Requirements.....	8
3.2 Software Requirements.....	9
4.Software Requirements Specification.....	10
4.1 Users	10
4.2 Functional Requirements.....	10
4.3 Non-Functional Requirements	12
5.System Design	13
5.1 Architecture Diagram	13
5.2 Block Diagram	16
6.Detailed Design	18
6.1 Process Flow.....	18
6.2 Use Case Diagram	20
6.3 Activity Diagram	22
7.Implementation.....	23
7.1 Screen Shots	23
References	33

Abstract

This project explores an innovative approach to human-computer interaction by combining eye and hand gestures to control the cursor pointer on a computer screen. The traditional input devices such as a mouse or touchpad are augmented with a system that utilizes both eye-tracking technology and hand gestures for a more intuitive and immersive user experience. The eye-tracking component enables precise cursor control based on the user's gaze, allowing for natural and fluid movement across the screen. Simultaneously, hand gestures enhance the system's versatility, providing users with an additional layer of interaction for tasks such as selecting, dragging, and dropping. The integration of these two input modalities creates a multifaceted and adaptable system that caters to a variety of user preferences and accessibility needs. This project not only explores the technical implementation of the combined eye and hand gesture control but also assesses its usability, efficiency, and potential applications in real-world scenarios. By merging these input methods, the project aims to offer a more inclusive and accessible computing experience. The results of this research contribute to the advancement of human-computer interaction technologies, paving the way for more natural and immersive computing experiences in the future.

1.Introduction

The world where computers are an essential part of our daily lives, interacting with them in a seamless and intuitive manner becomes increasingly important. Traditionally, we've relied on the familiar mouse and touchpad to navigate our screens. This project is all about! a new way for people to interact with computers, making it easier and more fun. By combining eye-tracking technology with hand gestures, we create a system that responds to where you look and how you move your hands. This means you can move the cursor around the screen just by looking at it, and perform actions like clicking or dragging with simple hand and eye gestures. Our goal is to make computer use more accessible for everyone, including those who may find traditional input devices challenging. This project not only focuses on making the technology work but also on making it easy and enjoyable for people to use. By letting your eyes and hands take the lead, we're looking towards creating a computer experience that feels friendly and easy to use.

1.1 Problem Scenario

Interacting with a computer cursor using the standard mouse or touchpad can be a bit tricky and uncomfortable for many users. Many individuals, regardless of age or ability, encounter difficulties in navigating the computer interface using standard input devices. This problem is further compounded by the potential for discomfort and repetitive strain injuries by using the traditional mouse. The conventional input devices might not provide the ease and comfort desired for an enjoyable computing experience. Recognizing these challenges, The project aims to address the difficulties associated with traditional cursor control methods, seeking to provide a more accessible and user-friendly alternative.

1.2 Proposed Solution

This project focuses on revolutionizing the way we interact with the computer cursor by eliminating the need for a conventional mouse or touchpad. Instead, we introduce a novel approach that allows users to control the cursor through intuitive hand and eye gestures. Picture being able to effortlessly guide the cursor across the screen by simply moving your hand or looking at where you want it to go. It's akin to playing with your computer using natural, instinctive movements, making the entire interaction process both enjoyable and accessible to everyone. This approach not only simplifies cursor control but also opens up new possibilities for a more intuitive and engaging computing experience, catering to a broad spectrum of users.

1.3 Scope

- The project may face limitations in scenarios where the user is operating in complete darkness. The absence of adequate ambient light could hinder the effectiveness of eye-tracking technology, leading to decreased performance.
- The success of the project relies on the quality of the camera used for eye and hand gesture tracking. Low-resolution or poorly calibrated cameras may compromise the precision and responsiveness of the system.

2.Literature Survey

The literature review chapter plays a critical role in laying the foundation for the current research by examining existing literature related to the problem domain. The primary aim is to investigate the research already conducted on similar problems, thereby gaining insights into the available data, materials, and findings. This extensive review is crucial for identifying gaps in the existing knowledge, which the present research aims to address. By dedicating sufficient time to this review, the researcher ensures a comprehensive understanding of the context, establishes a baseline of knowledge, and sets the stage for contributing new insights or solutions to the identified gaps in the literature.

2.1 Domain Survey

A comprehensive domain survey for systems delves into the realm of eye-tracking technology, hand gesture recognition, assistive technologies, and human-computer interaction. The exploration begins with an overview of the historical evolution of eye-tracking and hand gesture technology and its applications. The survey investigates the current landscape of computer accessibility, scrutinizing traditional input methods and emphasizing the need for alternative solutions. A detailed examination of the technological components involved in cursor control using eye and hand gestures systems follows, encompassing both hardware elements like cameras, and software aspects such as image processing algorithms and user interface design. The survey includes an evaluation of existing eye-controlled mouse systems, and hand gesture recognition system considering their features, accuracy, and user feedback. Challenges and limitations, both technical and ethical, are identified, highlighting the complexities involved in developing user-friendly and ethically sound solutions. The human-computer interaction and user experience aspects of the technology are explored, emphasizing considerations of usability, efficiency, and impact on daily life.

2.2 Existing Systems

2.2.1 Eye Tracking Mouse for Human Computer Interaction:

In the paper titled “Eye Tracking Mouse for Human-Computer Interaction,” presented at the 2013 E-Health and Bioengineering Conference (EHB) by authors Robert Gabriel Lupu, Florina Ungureanu, and Valentin Siriteanu, they introduced an innovative system utilizing eye-tracking techniques to facilitate human-computer interaction. The system, designed for mouse control based on eye movements, showcased advancements in reliable and hands-free computer interaction. However, a significant limitation was identified—specifically, the system ceased to function when any liquid, such as eyeliner or mascara, was present in the eyes. This constraint, particularly pertinent for female users who frequently use eye cosmetics, highlights a scenario in which the system's effectiveness is compromised. The research underscores the importance of addressing such limitations to enhance the robustness and adaptability of eye-tracking technology for diverse user scenarios in real-world applications.[1]

2.2.2 Face and Eye Tracking for Controlling Computer Functions:

In their 2014 paper titled "Face and Eye Tracking for Controlling Computer Functions," presented at the 11th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology (ECTI-CON), authors Chairat Kraichan and Suree Pumrin introduced a face and eye-controlled system developed using MATLAB. The system employed a webcam to enable mouse control through facial and eye movements, showcasing an innovative approach to hands-free computer interaction. However, a notable limitation was identified—the system's effective functionality was confined to a few centimeters, potentially restricting its practical usability. This work contributes to the field of human-computer interaction and highlights the importance of addressing operational constraints to enhance the system's adaptability for broader application.[2]

2.2.3 Pupil Centre Coordinates Detection Using the Circular Hough Transformation:

The paper titled "Pupil Centre Coordinates Detection Using the Circular Hough Transform Technique," published in the 2015 38th International Spring Seminar on Electronics Technology (ISSE) by Radu Gabriel Bozomitu, Alexandru Pasarica, Vlad Cehan, Cristian Rotariu, and Constantin Barabasa, introduces a system employing Hough Transform Techniques through a webcam for detecting pupil center coordinates. While demonstrating an effective method for pupil detection, the system faces a significant limitation—it operates at a non-real-time pace. The sequential process of capturing the body, face, eyes, and then the pupil results in substantial time delays, hindering the system's responsiveness. This drawback highlights the need for optimizations to transform the system into a real-time solution and enhance its practical applicability in various contexts.[3]

2.2.4 Cursor Control System Using Hand Gesture Recognition:

The Sneha U. Dudhane, Monika B. Gandhi, and Ashwini M. Patil in 2013 proposed a study on—Cursor Control System Using Hand Gesture Recognition. In this system, the disadvantages that the framed have to be stored first and then processed for detection which is much slower than what is required in real-time.[4]

2.2.5 A Real Time Hand Gesture Recognition System:

In 1990, Quam introduced an early hardware-based system; in this system, the user should wear a Data Glove. The proposed system by Quam although gives results of higher accuracy, but it is difficult to perform some of the gesture controls using the system. Data Gloves typically had sensors embedded in the glove to detect hand and finger movements. While they could provide a more accurate representation of hand gestures compared to some other methods, they did have limitations.[5]

2.3 Tools and Technologies

2.3.1 Python (v3.8.10):

Python is a high-level, interpreted programming language known for its readability, simplicity, and versatility. Created by Guido van Rossum, Python features a clean and concise syntax that prioritizes code readability and facilitates easy maintenance. Its support for multiple programming paradigms, including procedural, object-oriented, and functional programming, makes it adaptable to diverse applications. With an extensive standard library, Python minimizes the need for external dependencies. Being dynamically typed and interpreted, Python allows for flexible variable assignments and supports rapid development. The language boasts a large and active community, resulting in a robust ecosystem of libraries and frameworks. Python's cross-platform compatibility, popularity in data science and machine learning, and web development frameworks like Django contribute to its widespread use. This community-driven language, with its open development process and continuous improvement through the Python Enhancement Proposal (PEP) system, remains a preferred choice for developers ranging from beginners to experts across various domains.

2.3.2 OpenCV (v4.8.1):

OpenCV, or Open Source Computer Vision Library, is a widely adopted open-source software library designed for computer vision and machine learning applications. Developed in C++ and featuring bindings for Python and other languages, OpenCV provides an extensive set of tools for tasks such as image and video processing, object recognition, and machine learning. With support for parallel processing through OpenCL and CUDA, it enhances performance on GPUs. The library's versatility is evident in its applications, including camera calibration, object tracking, and contributed modules covering advanced topics. OpenCV's cross-platform compatibility, active community support, and modular structure make it a go-to resource for developers and researchers in academia and industry seeking robust solutions for a broad spectrum of computer vision challenges.

2.3.3 MediaPipe (v0.10.9):

MediaPipe, developed by Google, is an open-source framework designed for perceptual computing tasks, with a primary emphasis on computer vision. Offering a versatile suite of machine learning solutions, MediaPipe simplifies the development of applications involving real-time face detection, hand tracking, pose estimation, and more. Its key features include a robust hand tracking module, face detection with facial landmark estimation, pose estimation, and a holistic human understanding module that combines face, hand, and pose components. Developers can customize and integrate these components to build tailored pipelines for their applications. MediaPipe's cross-platform compatibility extends its deployment capabilities across desktops, mobile devices, and embedded systems. Benefiting from an active open-source community, MediaPipe stands out for its ease of use, flexibility, and the ability to empower developers, whether beginners or experts, in incorporating advanced computer vision capabilities into diverse projects with minimal effort.

2.3.4 Pyauto GUI (v0.9.54):

PyAutoGUI is a cross-platform Python module that provides functions to programmatically control the mouse and keyboard. It allows automation of tasks such as moving the mouse cursor, clicking, dragging, and sending keyboard input. PyAutoGUI is particularly useful for automating repetitive tasks, creating GUI testing scripts, and building simple automation workflows. The library supports multi-platform compatibility, making it suitable for Windows, macOS, and Linux environments. PyAutoGUI also includes features like taking screenshots and getting screen information. While it simplifies automation, developers should use it with caution to ensure that automated actions do not interfere with normal user interactions. Overall, PyAutoGUI serves as a handy tool for creating efficient and straightforward automation scripts using Python.

3. Hardware and Software Requirements

3.1 Hardware Requirements

3.1.1 Laptop or PC with web camera

The cursor pointer control using eye and hand gestures requires a standard laptop or desktop computer equipped with a built-in or external web camera. This camera serves as the primary input device for capturing and interpreting the user's eye movements.

3.1.2 RAM 8GB

The minimum of 8GB RAM is recommended to ensure the system's smooth operation. This memory capacity is essential for handling the real-time processing demands of eye-tracking and hand tracking algorithms, providing a responsive and efficient user experience.

3.1.3 256GB or More SSD/HDD

The system benefits from a storage capacity of 256GB or larger, utilizing either a Solid State Drive(SSD) or Hard Disk Drive(HDD). This storage space accommodates the necessary software components and ensures optimal performance of the system.

3.1.4 Intel i3 or AMD Ryzen 5 Processor

The system's processing power is supported by either an Intel i3 or AMD Ryzen 5 processor. These processors offer the computational capacity required for running eye-tracking and hand tracking algorithms and associated software, contributing to the system's responsiveness and accuracy.

3.2 Software Requirements

3.2.1 Windows Operating System

The cursor pointer control using eye and hand gestures system is designed to operate on the windows operating system. Compatibility with windows ensures a stable and supported environment for the software components and enhances user accessibility.

3.2.2 Python v3.8.10

The system relies on Python version 3.8.10 as a key programming language. Python serves as the foundation. This version is specified for compatibility and to ensure consistency in the software environment.

4. Software Requirements Specification

4.1 Users

- Individuals who are looking for a new and innovative way to interact with their computers beyond the conventional mouse or touchpad.
- Users who experience discomfort or strain using traditional input devices and are looking for more ergonomic alternatives to enhance their computing experience.
- The software is designed to be user-friendly to individuals with varying levels of computer experience. The aim is to create an intuitive interface for seamless interaction.

4.2 Functional Requirements

4.2.1 Eye Tracking:

An integral feature of the system is its ability to accurately track the movement of both eyes. It delivers real-time updates on the position and gaze direction of the eyes, ensuring a responsive and dynamic eye-controlled interface that aligns with the user's intentions and actions.

4.2.2 Hand Gesture Recognition:

The Hand gesture recognition involves capturing and interpreting movements and positions of the user's hand. Cameras technologies can be used to track hand movements accurately. Gesture recognition algorithms analyze the data to identify specific hand gestures.

4.2.3 Click Actions:

Users have the capability to perform click actions through eye blinks and through hand gestures. This innovative feature translates gestures into meaningful interactions, allowing users to execute click commands without physical contact, thereby providing a seamless and accessible means of interacting with the system.

4.2.4 Scrolling: Scrolling in the cursor control using eye and hand gestures typically involves navigating content on a display, such as scrolling through a document, webpage, or other graphical user interface elements. Specific gesture can be used to perform scrolling action.

4.2.5 Volume Control: Integrating volume control into a system that uses hand gestures for interaction involves recognizing specific gestures or movements associated with adjusting audio volume.

4.2.6 Drag and Drop: Integrating drag and drop functionality using hand gestures involves recognizing specific gestures or movements associated with dragging and dropping objects on a display.

4.3 Non-Functional Requirements

4.3.1.Performance:

The system prioritizes performance by responding to user eye movements with minimal latency, ensuring a seamless and responsive user experience. High accuracy in eye location tracking further enhances the precision and reliability of the system, contributing to an efficient and effective eye-controlled interface.

4.3.2.Usability:

Usability is a key focus of the system, emphasizing ease of learning and use, even for individuals with limited computer experience. The design aims to create an intuitive and user-friendly interface, facilitating accessibility and inclusivity for a diverse user base.

4.3.3.Reliability:

The system is engineered for reliability, demonstrating consistent operation over extended periods of use. It minimizes errors or crashes, ensuring a stable and trustworthy user experience. Additionally, the system is designed to handle errors gracefully, allowing for recovery from failures and maintaining overall system robustness.

4.3.4.Adaptability to Different Conditions:

The system ensures adaptability to various eye conditions, ensuring optimal effectiveness for users with different eye shapes and sizes. The system is capable of accurately tracking eye movements, regardless of variations in eye shapes and sizes among different users.

5.System Design

5.1 Architecture Diagram

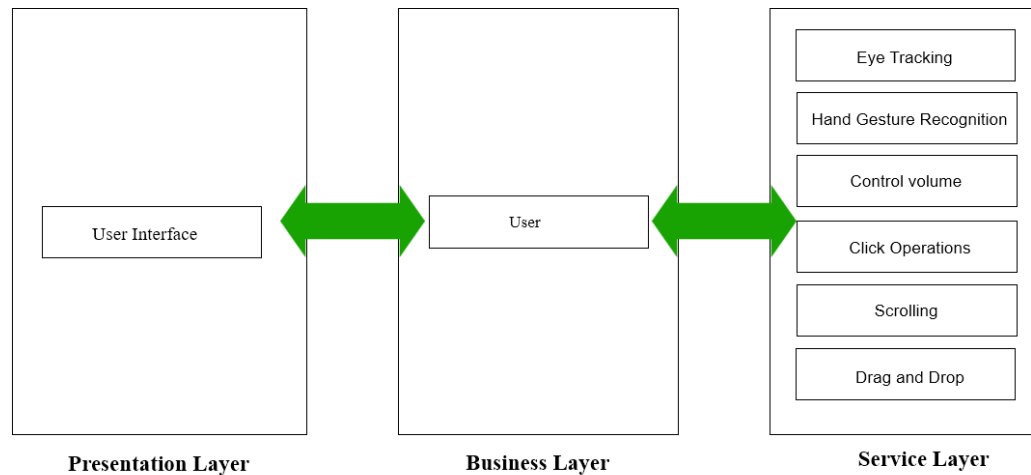


Figure 5.1 Cursor Pointer Control Using Eye and Hand Gestures System's Architecture Diagram

5.1.1 User:

In this system, users seamlessly provide input through their eyes and hands. The system captures and interprets the user's natural eye and hand gestures, translating them into precise on-screen cursor motions and facilitating mouse-free interactions.

5.1.2 Presentation Layer:

The Presentation Layer is the user-facing component of the cursor control using eye and hand gesture system. It encompasses the graphical user interface (GUI) elements, providing users with an intuitive and visually effective platform to interact with the system. This layer includes on-screen cursor representation focusing on delivering a user-friendly experience.

5.1.3 Business layer:

The business layer also known as the logic layer, is the layer responsible for handling the core business logic and functionality of the system. In this layer, the system's fundamental operations, algorithms, and data processing occur.

5.1.3.1 Hand Gestures Recognition Module:

The users can control the cursor by making hand gestures in the air. The system captures the movements and translates them into on-screen actions. Hand gestures can be used to simulate mouse clicks and perform selection. Gestures like swipe or pinch-to-zoom can be employed for scrolling or zooming in and out on content.

5.1.3.2 Eye tracking Module:

Users can control the cursor on the screen by simply moving their eyes. Eye-tracking technology detects the direction of gaze and translates it into cursor movement.

5.1.3.3 Click Operation Module:

The Click functionality in cursor pointer control using eye and hand gestures refers to the action of simulating a mouse click without physically pressing a button. This is done by both hand gesture or by just blinking the eye.

5.1.3.4 Scrolling Module:

The Scrolling functionality in cursor pointer control using eye and hand gestures allows users to scrolling through content, such as web pages or documents, without physically interacting with a traditional scrolling device like a mouse wheel or touchpad. The users just have to use their hand and specified gesture to scroll.

5.1.3.5 Drag and Drop Module:

The Drag and drop functionality allows users to interact with digital content by selecting, moving, and dropping items on a screen. When using hand gestures for drag and drop, the goal is to simulate these actions without the need for physical devices like a mouse.

5.1.3.5 Volume Control Module:

Volume control using hand gestures involves manipulating audio levels without the need for physical buttons or sliders.

5.2 Block Diagram

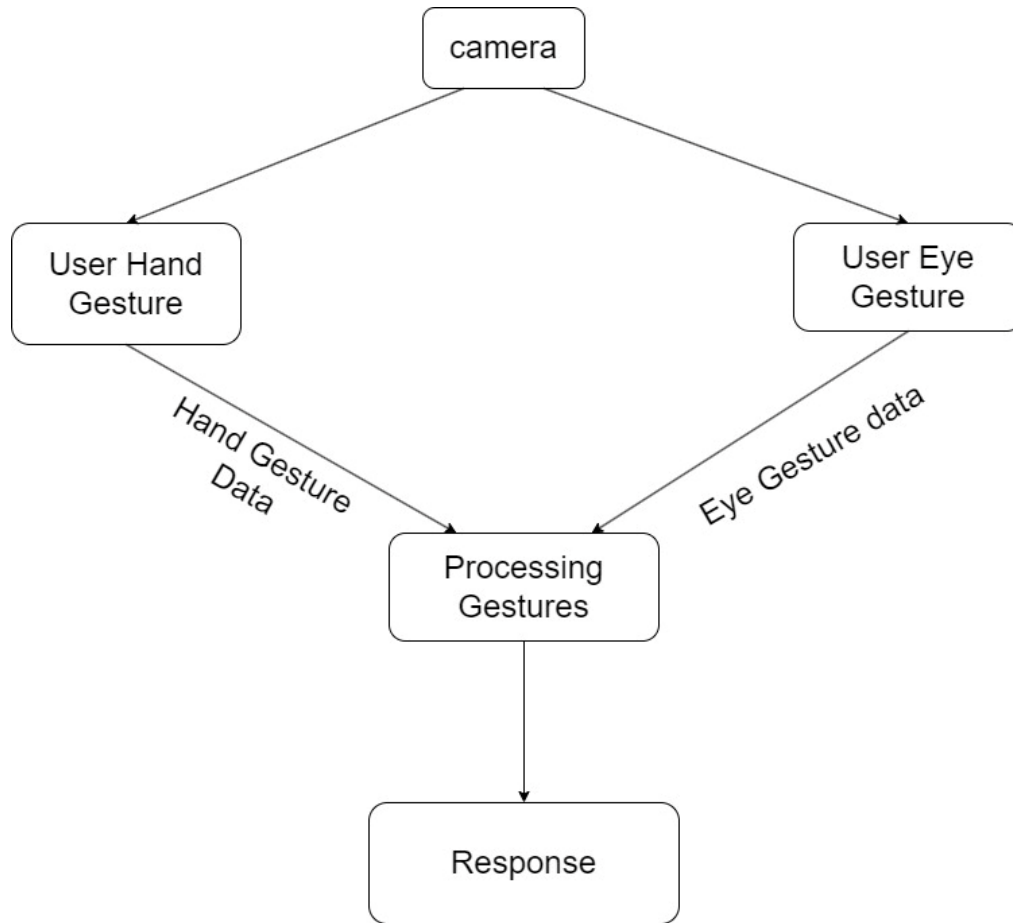


Figure 5.2 Cursor Pointer Control Using Eye and Hand Gestures System's Block Diagram

The diagram shows a system for controlling a computer cursor using both hand gestures and eye tracking. The system consists of the following components:

- **User Hand:** This is the hand of the user who will be making gestures to control the cursor. The hand gestures will be captured by a camera or other sensor.
- **User Eye:** This is the user's eye that will be tracked to determine where the user is looking on the screen. Eye tracking can be done using a specialized camera or sensor.
- **Hand Gesture Data / Eye Tracking Data:** The hand gesture data and eye tracking data are the inputs to the system. The hand gesture data is captured by the camera or sensor and processed to determine the specific gesture being made. The eye tracking data is used to determine where the user is looking on the screen.

- **Processing Gestures:** This component of the system processes the hand gesture data to determine the specific gesture being made. This may involve pattern recognition algorithms or other techniques to identify the gesture.
- **Control Signals:** Once the gesture has been identified, control signals are generated to move the cursor on the screen. The control signals may be sent directly to the computer or may be processed further to refine the cursor movement.
- **Computer:** The computer receives the control signals and moves the cursor accordingly.
- **Operate Cursor:** The final component of the system is the cursor itself, which is moved on the screen in response to the hand gestures and eye tracking data.

6.Detailed Design

6.1 Process Flow

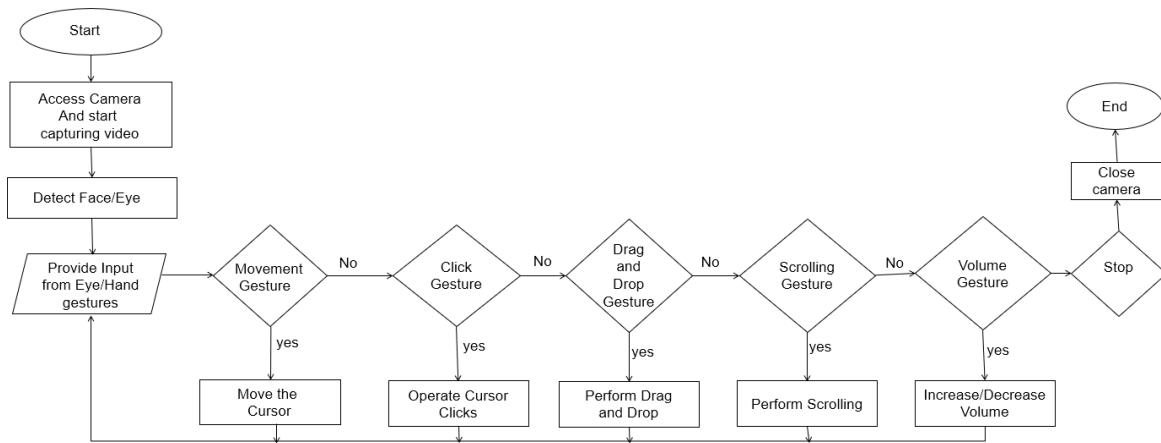


Figure 1 Cursor Pointer Control Using Eye and Hand Gestures System's Process Flow Diagram

This process flow diagram shows the steps involved in a system for controlling a computer cursor and performing various actions like clicking, drag-and-drop, scrolling, and volume control.

- **Start:** The process begins with starting the system and accessing the camera to capture video.
- **Detect Hand/Eye:** The system then detects the user's hand and eyes using computer vision techniques.
- **Provide Input from Eye/Hand Gestures:** The system tracks the user's eye and hand movements to provide input for controlling the cursor and performing various actions.
- **Movement Gesture:** If the user moves their eyes or hands in a certain way, the system moves the cursor to the corresponding location on the screen.
- **Click Gesture:** If the user performs a click gesture (e.g., by blinking their eyes or making a hand gesture), the system performs a click action on the current cursor location.
- **Drag and Drop Gesture:** If the user performs a drag-and-drop gesture through hands the system performs a drag-and-drop action on the corresponding screen elements.

- **Scrolling Gesture:** If the user performs a scrolling gesture, the system performs a scrolling action on the current screen.
- **Volume Gesture:** If the user performs a volume gesture, the system increases or decreases the volume accordingly.
- **End:** The process ends with closing the camera and stopping the system

6.2 Use Case Diagram

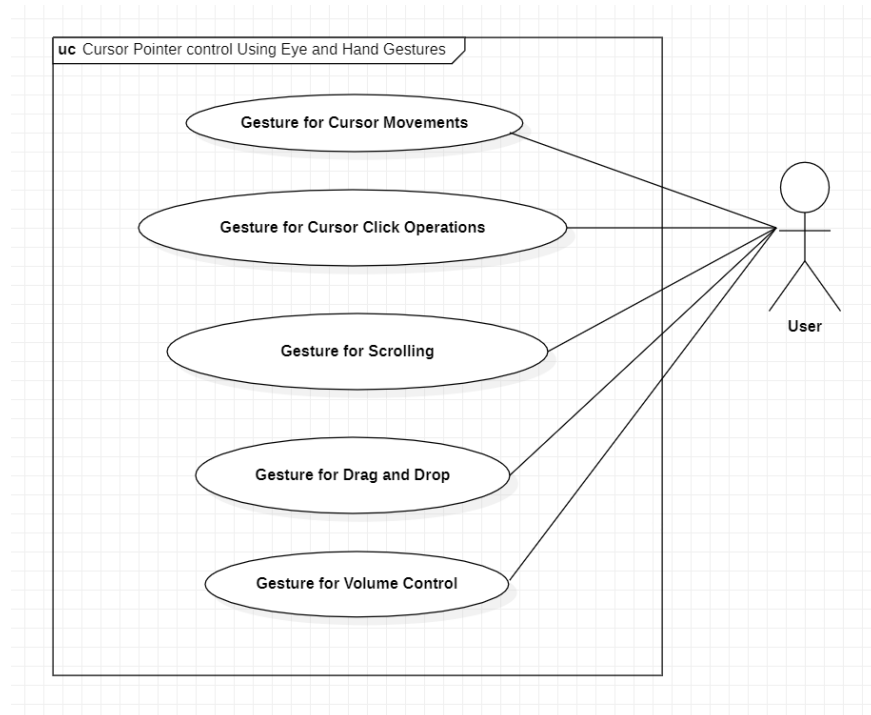


Figure 2 Cursor Pointer Control Using Eye and Hand Gestures System's Use Case Diagram

The use case diagram for the cursor pointer control using eye and hand gestures system visually represents the interaction scenarios between users and the system. The primary actor is the user.

6.2.1 Actors:

The primary actor represents the user interacting with the system. The person who will be using the system to control the cursor pointer.

6.2.2 Use Cases:

- **Gesture for Cursor Movements:** A use case where the user can move the cursor pointer by performing a specific gesture with their eyes or hands.
- **Gesture for Cursor Click Operations:** A use case where the user can perform click operations (e.g. left-click, right-click) by performing a specific gesture with their eyes or hands.
- **Gesture for Scrolling:** A use case where the user can scroll through a document or webpage by performing a specific gesture with their hands.

- Gesture for Drag and Drop: A use case where the user can drag and drop items by performing a specific gesture with their hands.
- Gesture for Volume Control: A use case where the user can control the volume of the system by performing a specific gesture with their hands.

6.3 Activity Diagram

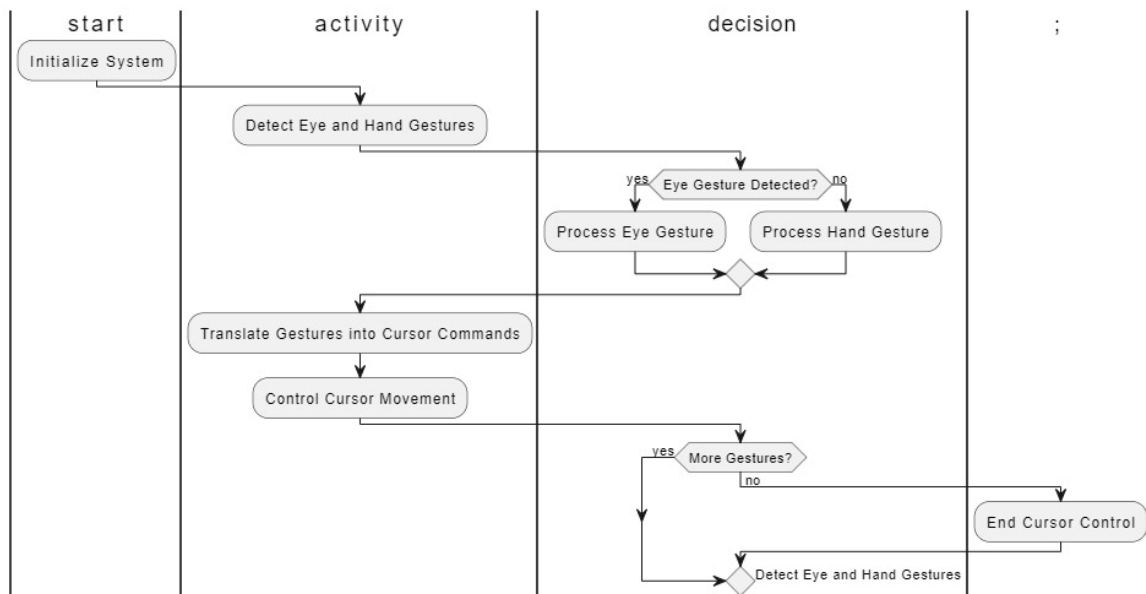
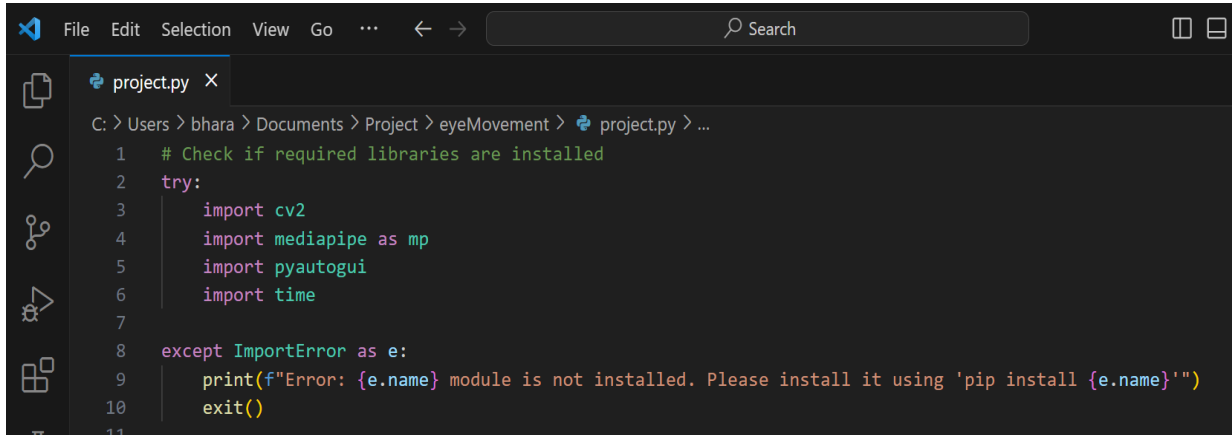


Figure 6.3 Cursor pointer Control Using Eye and Hand Gesture System's Activity Diagram

- **Initialize System:** The process begins with initializing the system.
- **Detect Eye and Hand Gestures:** The system continuously monitors and detects both eye and hand gestures.
- **Decision Point:** The diagram branches based on whether an eye gesture is detected. If an eye gesture is detected, the system proceeds to Process Eye Gesture. If no eye gesture is detected, the system proceeds to Process Hand Gesture.
- **Process Eye Gesture:** The system processes the detected eye gesture.
- **Process Hand Gesture:** The system processes the detected hand gesture.
- **Translate Gestures into Cursor Commands:** Regardless of the type of gesture (eye or hand), the system translates the detected gestures into cursor commands.
- **Control Cursor Movement:** The translated cursor commands are used to control the movement of the cursor.
- **Decision Point:** The diagram includes a decision point to check if there are more gestures to be detected. If there are more gestures, the system goes back to Detect Eye and Hand Gestures. If no more gestures are detected, the process proceeds to the end.
- **End Cursor Control:** The diagram concludes with the end of the cursor control process.

7.Implementation

7.1 Screen Shots

A screenshot of a code editor window with a dark theme. The window title is 'project.py'. The menu bar includes 'File', 'Edit', 'Selection', 'View', 'Go', and a search bar. The file explorer on the left shows the path 'C: > Users > bhara > Documents > Project > eyeMovement > project.py'. The code in the editor is a Python script that checks for the installation of required libraries. It uses a try-except block to attempt importing cv2, mediapipe, pyautogui, and time. If an ImportError occurs, it prints an error message and exits. The code is as follows:

```
1 # Check if required libraries are installed
2 try:
3     import cv2
4     import mediapipe as mp
5     import pyautogui
6     import time
7
8 except ImportError as e:
9     print(f"Error: {e.name} module is not installed. Please install it using 'pip install {e.name}'")
10    exit()
11
```

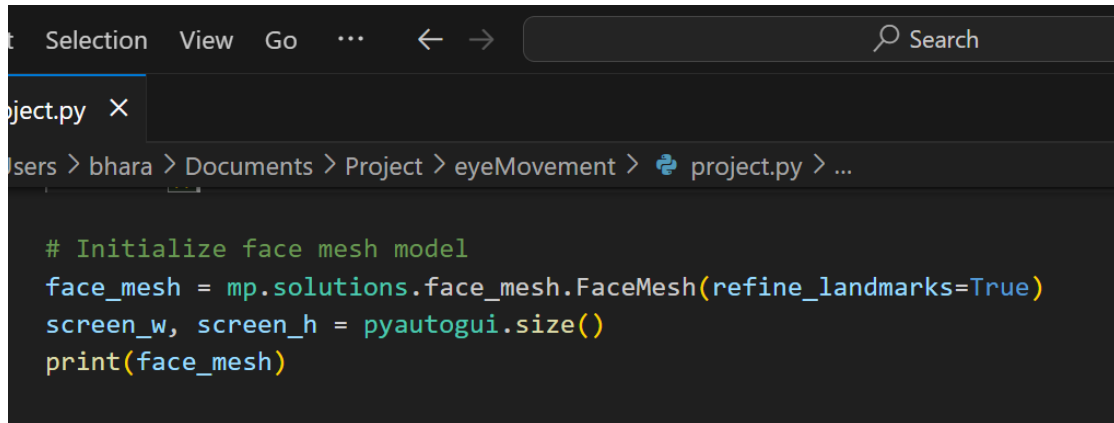
Figure 7.1.1 Importing Packages

This Python script utilizes several external libraries for computer vision and automation purposes. It begins by attempting to import the necessary modules including OpenCV, Mediapipe, PyAutoGUI and the time module. These libraries are commonly employed for image processing, hand tracking, automation of mouse functions, and time-related operations. The script is designed to raise an error and prompt the user to install any missing modules using pip if they are not already installed. Once the required libraries are successfully imported, they can be leveraged for a variety of applications, such as hand gesture recognition or eye tracking, depending on the specific implementation of the script. The inclusion of error handling ensures that users are informed about missing dependencies, contributing to a smoother execution of the script.

```
15
16 #starting to capture the video
17 try:
18     cam = cv2.VideoCapture(0)
19
20     #checks if the camera is opened and if not generates error and code exits
21     if not cam.isOpened():
22         raise Exception("Could not open camera.")
23
24 except Exception as e:
25     print(f"Error: {e}")
26     exit()
27
```

Figure 7.1.2 Accessing computer's camera

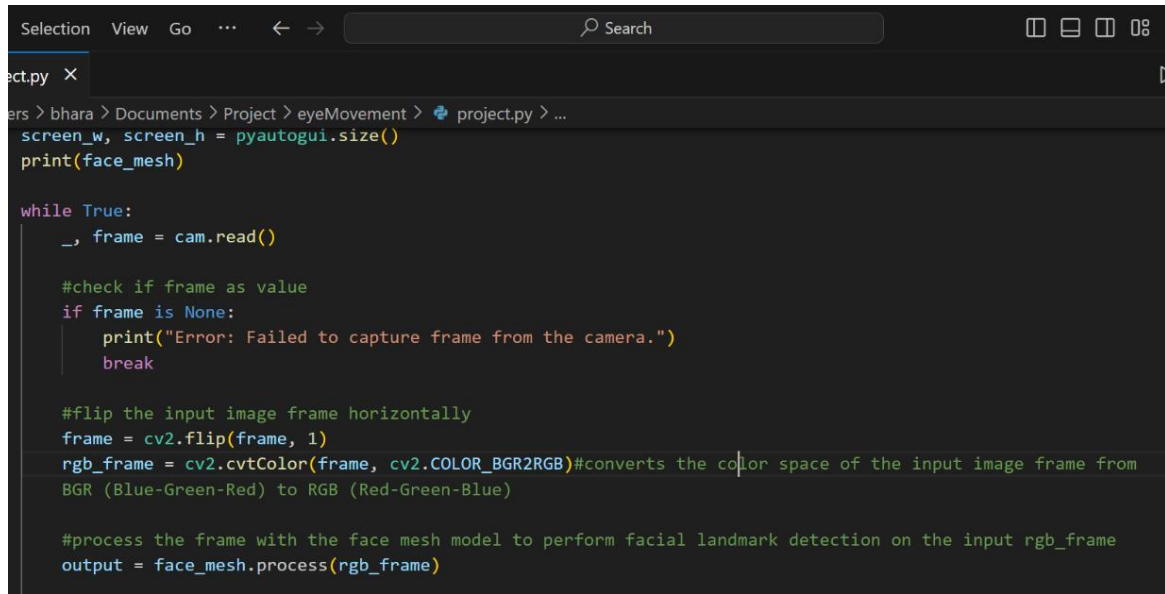
This Python code snippet revolves around the initialization and error-handling procedures for accessing the computer's camera using the OpenCV library. It starts by creating a VideoCapture object named cam and attempts to open the default camera using the 'cv2.VideoCapture(0)' function. Following this, the script includes a crucial verification step to check if the camera is successfully opened by inspecting the return value of 'cam.isOpened()'. In the event that the camera fails to open, an exception is raised, specifically an instance of the generic 'Exception' class. The raised exception contains a custom error message stating, "Could not open camera". Subsequently, the script catches and handles this exception, printing the detailed error message to the console. Finally, the script invokes the 'exit()' function, gracefully terminating the code execution. This meticulous error-handling mechanism ensures that users are promptly informed if there is any issue with camera access, contributing to a more robust and user-friendly script execution.

A screenshot of a code editor window. The title bar shows 'Selection View Go ...' and a search icon. The file name is 'project.py'. The breadcrumb path is 'Users > bhara > Documents > Project > eyeMovement > project.py > ...'. The code is as follows:

```
# Initialize face mesh model
face_mesh = mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)
screen_w, screen_h = pyautogui.size()
print(face_mesh)
```

Figure 3.1.3 Initializing Face Mesh Model

In this Python code snippet, a face mesh model is initialized using the Mediapipe library 'mp.solutions.face_mesh.FaceMesh'. The 'refine_landmarks' parameter is set to 'True', indicating that the face landmarks detected by the model should be refined for increased accuracy. Additionally, the code retrieves the screen width and height using 'pyautogui.size()' and assigns these values to 'screen_w' and 'screen_h'. Finally, the script prints information about the initialized face mesh model, providing insights into its configuration and settings. This segment of code sets the foundation for further face mesh analysis, potentially for applications such as facial feature tracking or gesture recognition based on facial landmarks.

A screenshot of a code editor window with a dark theme. The editor shows a Python script for capturing frames from a camera. The code includes a 'while True' loop that reads a frame from a camera object 'cam'. It checks if the frame is None, prints an error message, and breaks the loop if so. The frame is then horizontally flipped using 'cv2.flip()' and converted from BGR to RGB color space using 'cv2.cvtColor()'. Finally, the frame is processed by a 'face_mesh' object to perform facial landmark detection, with the result stored in 'output'.

```
ect.py x
ers > bhara > Documents > Project > eyeMovement > project.py > ...
screen_w, screen_h = pyautogui.size()
print(face_mesh)

while True:
    _, frame = cam.read()

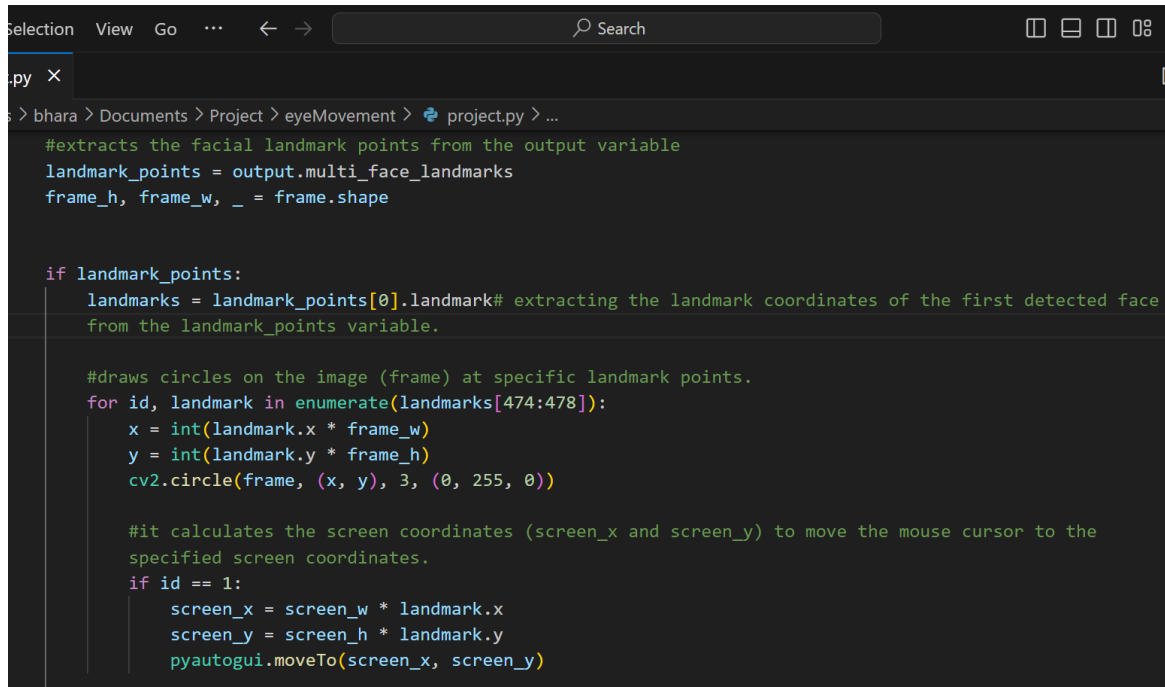
    #check if frame as value
    if frame is None:
        print("Error: Failed to capture frame from the camera.")
        break

    #flip the input image frame horizontally
    frame = cv2.flip(frame, 1)
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)#converts the color space of the input image frame from
    BGR (Blue-Green-Red) to RGB (Red-Green-Blue)

    #process the frame with the face mesh model to perform facial landmark detection on the input rgb_frame
    output = face_mesh.process(rgb_frame)
```

Figure 7.1.4 Capturing Each Frames

In this Python code segment, a continuous loop ‘while True’ is implemented to capture frames from the camera using OpenCV ‘cam.read()’. The underscore ‘_’ is used as a convention to discard the first value returned by ‘cam.read()’, which typically represents whether the read operation was successful. The script then checks if the captured frame has a valid value, and if not, it prints an error message and breaks out of the loop. Next, the captured frame is horizontally flipped using ‘cv2.flip()’ to account for potential mirroring effects in camera input. The color space of the frame is converted from BGR (Blue-Green-Red) to RGB (Red-Green-Blue) using ‘cv2.cvtColor()’. The transformed RGB frame is then processed using the initialized face mesh model ‘face_mesh.process()’, aiming to perform facial landmark detection. The results of this processing are stored in the variable ‘output’. This loop structure ensures a continuous flow of frame processing, potentially for real-time applications involving facial feature tracking or analysis based on the detected facial landmarks.



```
Selection View Go ... ← → Search
.py x
> bhara > Documents > Project > eyeMovement > project.py > ...

#extracts the facial landmark points from the output variable
landmark_points = output.multi_face_landmarks
frame_h, frame_w, _ = frame.shape

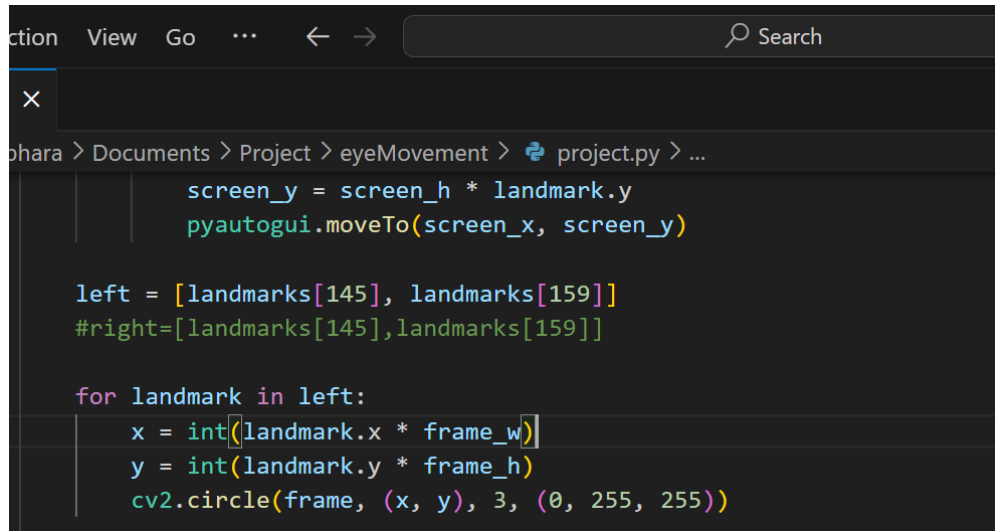
if landmark_points:
    landmarks = landmark_points[0].landmark# extracting the landmark coordinates of the first detected face
    from the landmark_points variable.

    #draws circles on the image (frame) at specific landmark points.
    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))

    #it calculates the screen coordinates (screen_x and screen_y) to move the mouse cursor to the
    specified screen coordinates.
    if id == 1:
        screen_x = screen_w * landmark.x
        screen_y = screen_h * landmark.y
        pyautogui.moveTo(screen_x, screen_y)
```

Figure 7.1.5 Extracting landmarks of Right Eye

In this part of the Python script, the code checks if there are landmark points detected in the processed frame by examining the 'landmark_points' variable. If landmark points are present, the script extracts the landmark coordinates of the first detected face from 'landmark_points'. Subsequently, the code iterates through a subset of the landmark points (in this case, landmarks 474 to 477) and draws circles on the original frame ('frame') at these specific landmark locations. The circles are visual indicators on the frame to represent the detected facial landmarks. Additionally, the script calculates the screen coordinates ('screen_x' and 'screen_y') based on the landmark coordinates. This information is then used to move the mouse cursor to the specified screen coordinates using the 'pyautogui.moveTo()' function. Specifically, when the iteration reaches the second landmark (id == 1), the cursor is moved to the calculated screen coordinates. This segment of code demonstrates the integration of facial landmark information with mouse cursor control, potentially enabling functionalities like controlling the cursor with facial gestures or movements.

A screenshot of a code editor window with a dark theme. The editor shows a Python script for eye movement tracking. The code includes a search bar at the top, a file explorer on the left, and a main code area. The code defines a list 'left' with two landmarks, calculates their screen coordinates, and draws circles on a frame. The file path in the explorer is 'phara > Documents > Project > eyeMovement > project.py > ...'.

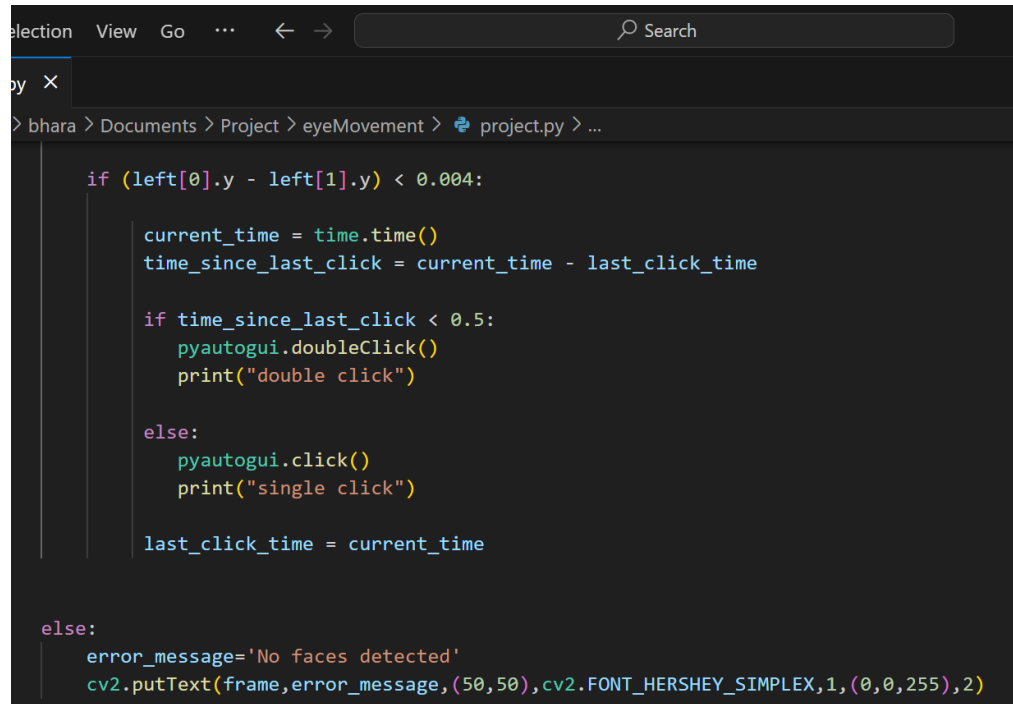
```
screen_y = screen_h * landmark.y
pyautogui.moveTo(screen_x, screen_y)

left = [landmarks[145], landmarks[159]]
#right=[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))
```

Figure 7.1.6 Extracting Left Eye

In this part of the Python script, a region of interest is defined for the left side of the face by selecting specific landmarks. Specifically, the script creates a list 'left' containing two landmarks (landmarks[145] and landmarks[159]) that correspond to points on the left side of the face. The code then iterates through these left-side landmarks and draws circles on the original frame ('frame') at the calculated (x, y) coordinates. The circles serve as visual markers to highlight the selected landmarks on the left side of the face. This type of analysis could be utilized for facial feature tracking or for implementing interactions based on specific facial gestures or expressions associated with the left side of the face. It's worth noting that the code snippet provided here is a part of a larger script that likely involves further processing and analysis of facial landmarks for specific applications.

A screenshot of a code editor window with a dark theme. The editor shows a Python script with logic for detecting double and single clicks based on facial landmarks. The code includes comments and uses the 'time' and 'pyautogui' modules. The file path in the breadcrumb is 'bhara > Documents > Project > eyeMovement > project.py'.

```
selection View Go ... ← → Search  
by X  
> bhara > Documents > Project > eyeMovement > project.py > ...  
  
    if (left[0].y - left[1].y) < 0.004:  
        current_time = time.time()  
        time_since_last_click = current_time - last_click_time  
  
        if time_since_last_click < 0.5:  
            pyautogui.doubleClick()  
            print("double click")  
  
        else:  
            pyautogui.click()  
            print("single click")  
  
        last_click_time = current_time  
  
    else:  
        error_message='No faces detected'  
        cv2.putText(frame,error_message,(50,50),cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)
```

Figure 7.1.7 Click Operations

In this portion of the Python script, the code performs additional processing based on the relative vertical position of the selected left-side facial landmarks. Specifically, it checks if the difference in the y-coordinates of the two landmarks (`left[0].y` and `left[1].y`) is less than 0.004. If this condition is met, it implies that the left side of the face is relatively flat or level. Upon detecting a relatively flat left side of the face, the script calculates the time elapsed since the last click operation using the 'time' module. If the time since the last click is less than 0.5 seconds, the script executes a double-click operation using '`pyautogui.doubleClick()`'. Otherwise, a single click is performed using '`pyautogui.click()`'. The current time is then stored as the '`last_click_time`', ensuring accurate tracking of the time interval between consecutive clicks. Additionally, the script prints a message to the console indicating whether a single or double click was executed. If no faces are detected (the 'else' statement), an error message is displayed on the frame using '`cv2.putText()`', indicating that no faces were found. This segment of the script suggests that the facial landmarks on the left side of the face are being utilized to trigger mouse click events, potentially enabling hands-free interaction based on facial gestures or expressions.

```
exit_message = "Press 'Esc' to exit"
cv2.putText(frame, exit_message, (50, 100), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

cv2.imshow('Eye Controlled Mouse', frame)
key=cv2.waitKey(1)

if key == 27:
    break

#releasing camera
cam.release()
cv2.destroyAllWindows()
```

Figure 7.1.8 Releasing the camera

In this final part of the Python script, the code adds some user interface elements and handles the exit mechanism. It starts by displaying an exit message at the specified location on the frame using 'cv2.putText()'. The message instructs the user to press the 'Esc' key to exit the application. Subsequently, the script uses 'cv2.imshow()' to display the processed frame with the added elements, creating a window titled 'Eye Controlled Mouse'. The 'cv2.waitKey(1)' function waits for a key event, and if the key pressed is 'Esc' (27 in ASCII), the script breaks out of the continuous loop using the 'break' statement. Finally, the script releases the camera using 'cam.release()' to free up system resources and closes all OpenCV windows with 'cv2.destroyAllWindows()'. This part of the script manages the display of information on the frame, creating a simple user interface, and provides a clear way for the user to exit the application using the 'Esc' key. The overall structure ensures a smooth user experience while interacting with the eye-controlled mouse application.

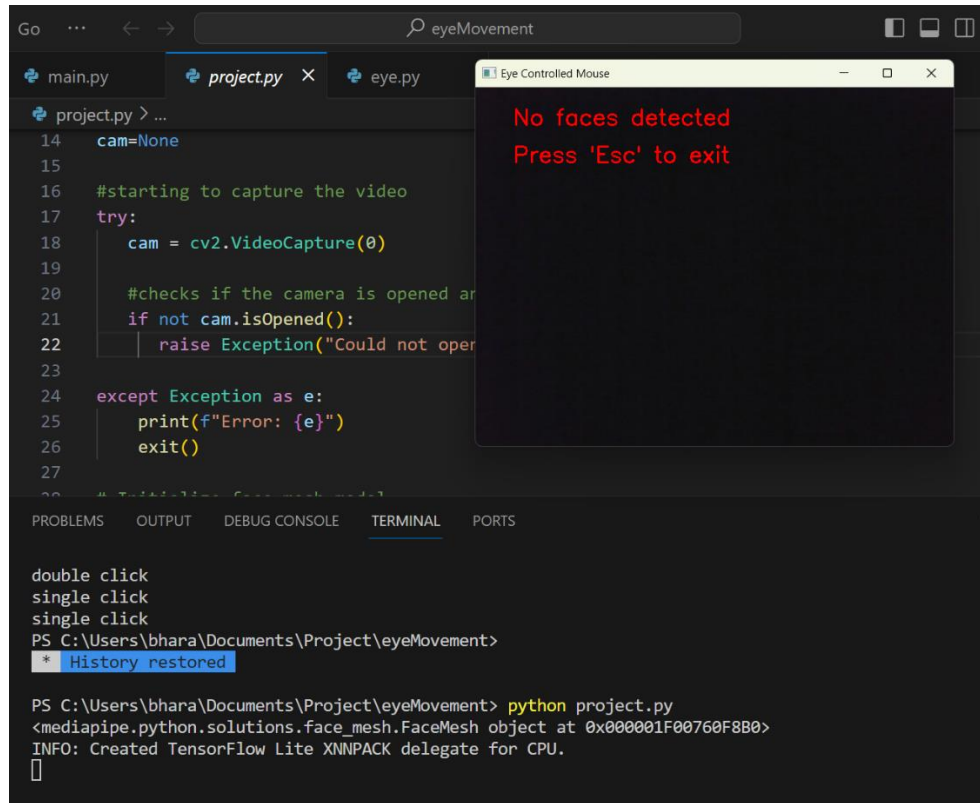


Figure 7.1.9 No Faces Detected Error

The eye-controlled mouse system begins by accessing the computer's camera and attempting to detect a face using OpenCV and a face mesh model from the Mediapipe library. If a face is successfully detected, the system proceeds to extract facial landmarks, particularly focusing on the left side of the face. The vertical position of selected landmarks is analyzed, and if the left side is relatively flat, indicating a specific facial gesture or expression, the system calculates the time elapsed since the last mouse click. Based on this information, the system performs either a single or double click using the PyAutoGUI library, simulating mouse click operations. In case no face is detected, the system displays an error message on the captured frame. The user is provided with an exit message, prompting them to press the 'Esc' key to terminate the application. The entire process is implemented within a continuous loop, ensuring real-time monitoring and interaction. This eye-controlled mouse system integrates computer vision, facial landmark detection, and mouse control functionalities to offer a hands-free and potentially more accessible means of computer interaction based on facial expressions.

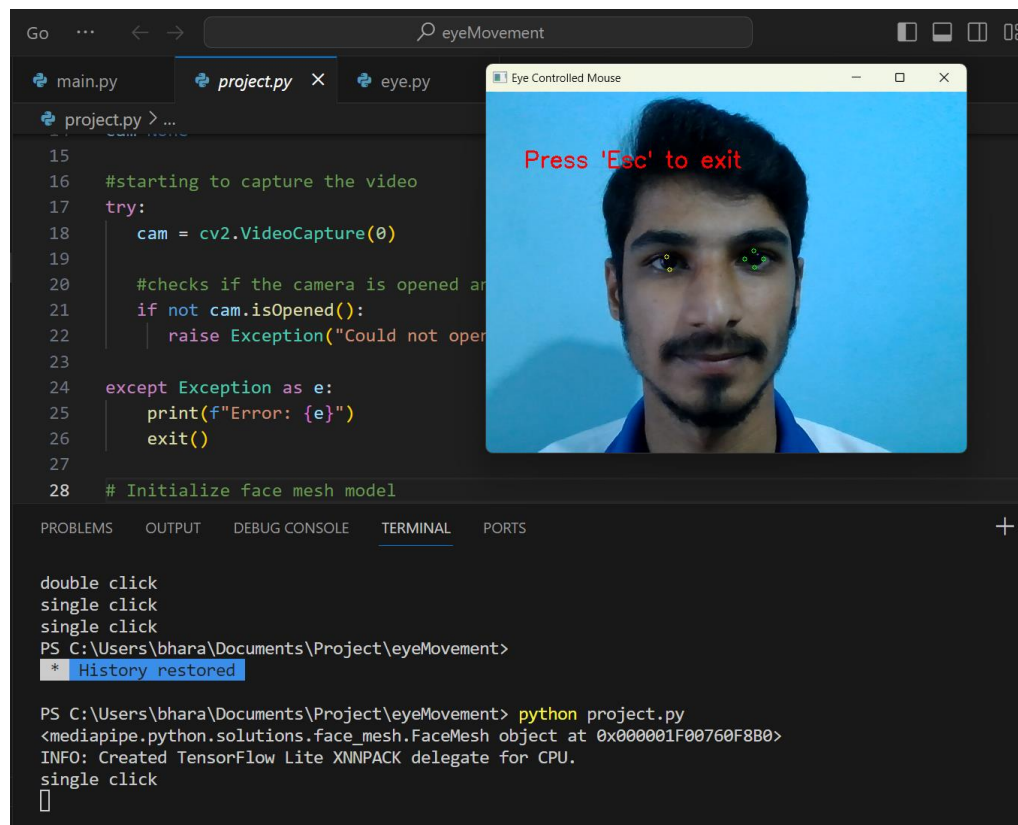


Figure 7.1.10 Detecting Eye Locations

The eye-controlled mouse system incorporates cursor movements and click operations based on facial landmarks. After successfully detecting a face and extracting facial landmarks, the system focuses on specific landmarks associated with the left side of the face. It tracks the vertical position of these landmarks and, if a certain facial expression is identified (indicating a relatively flat left side of the face), the system calculates the time elapsed since the last mouse click. Depending on this timing information, the system executes either a single or double mouse click using the PyAutoGUI library. To visualize and analyze the facial landmark movements, circles are drawn on the captured frame at the selected landmark locations. This not only provides visual feedback on the detected facial features but also aids in understanding the system's decision-making process regarding mouse click actions. The implementation integrates computer vision, facial gesture recognition, and mouse control functionalities, offering users a hands-free and potentially more intuitive method of interacting with the computer through facial expressions.

References

1. [“Eye tracking mouse for human-computer interaction”](https://ieeexplore.ieee.org/document/6707244/)
<https://ieeexplore.ieee.org/document/6707244/>(2013)
2. [“Face and eye tracking for controlling computer functions”](https://ieeexplore.ieee.org/document/6839834/)
<https://ieeexplore.ieee.org/document/6839834/>(2014)
3. [“Pupil center coordinates detection using the circular Hough transform technique”](https://ieeexplore.ieee.org/document/7248041/)
<https://ieeexplore.ieee.org/document/7248041/>(2015)
4. [“An image-based eye controlled assistive system for paralytic patients”](https://ieeexplore.ieee.org/document/8066549/)
<https://ieeexplore.ieee.org/document/8066549/>(2017)
5. [“An eye tracking algorithm based on Hough transform”](https://ieeexplore.ieee.org/document/8408915/)
<https://ieeexplore.ieee.org/document/8408915/>(2018)