A project report on

EYE-CONTROLLED MOUSE SYSTEM USING FACIAL LANDMARKS AND GESTURE RECOGNITION

Submitted in partial fulfillment for the award of the degree of

Master of Technology in Computer Science and Engineering with Specialization in Big Data Analytics

by

SRUDHAN EERLA (24MCB1002)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

November,2025

EYE-CONTROLLED MOUSE SYSTEM USING FACIAL LANDMARKS AND GESTURE RECOGNITION

Submitted in partial fulfillment for the award of the degree of

Master of Technology in Computer Science and Engineering with Specialization in Big Data Analytics

by

SRUDHAN EERLA (24MCB1002)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

November, 2025

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I hereby declare that the thesis entitled "EYE-CONTROLLED MOUSE SYSTEM USING FACIAL LANDMARKS AND GESTURE RECOGNITION" submitted by SRUDHAN EERLA (24MCB1002), for the award of the degree of Master of Technology in Computer Science and Engineering with Specialization in Big Data Analytics, Vellore Institute of Technology, Chennai is a record of bonafide work carried out by me under the supervision of Dr. Modigari Narendra.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: Signature of the Candidate

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled **"EYE-CONTROLLED MOUSE SYSTEM USING FACIAL LANDMARKS AND GESTURE RECOGNITION"** is prepared and submitted by **Srudhan Eerla (24MCB1002)** to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of **Master of Technology in Computer Science and Engineering with Specialization in Big Data Analytics** programme is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide

Name: Dr. Deepa Nivethika.

Date:

Signature of the Examiner	Signature of the Examiner
Name:	Name:
Date:	Date:

Approved by the Head of Department,
Master of Technology in Computer Science and Engineering with Specialization in Big Data Analytics

Name: **Dr K Sathyarajasekaran**
Date:

ABSTRACT

This project aims to develop an intelligent Human–Computer Interaction (HCI) system that enables seamless, hands-free control of computer functions through facial gestures. This system is primarily intended to assist individuals with motor impairments while also offering an alternative mode of interaction for general users. It utilizes a standard webcam to capture live video input, which is processed using Dlib's facial landmark predictor in combination with OpenCV to accurately detect and track key facial features such as the eyes, mouth, and nose. By analyzing geometric relationships among these landmarks, the system identifies specific facial expressions such as winks, blinks, and mouth openings and converts them into corresponding mouse actions like cursor navigation, left or right clicks, and scrolling through the PyAutoGUI library.

The proposed architecture integrates modules for facial detection, landmark localization, and gesture-to-command mapping to ensure smooth and responsive performance in real time without relying on external sensors or hardware. Experimental outcomes indicate that the system provides an efficient, low-cost, and accessible solution for hands-free computing. Overall, this project demonstrates the capability of computer vision and facial-analysis techniques to enhance assistive technologies, broaden digital accessibility, and advance natural human–computer interaction paradigms.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Deepa Nivethika, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part to work with an intellectual and expert in the field of Machine Learning.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor, Dr. T. Thyagarajan Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Viswanathan V, Dean, Dr. Nithyanandam P, Associate Dean, Dr. Suganya G, Associate Dean, Dr. SweetlinHemalatha C, Associate Dean, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant state, I express ingeniously my whole-hearted thanks to Dr. K Sathyarajasekaran, Head of the Department, Master of Technology in Computer Science and Engineering with Specialization in Big Data Analytics and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staffs at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: **Srudhan Eerla**

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS

AI	ARTIFICIAL INTELLIGENCE
API	APPLICATION PROGRAMMING INTERFACE
BGR	BLUE GREEN RED (COLOR FORMAT USED IN IMAGES)
CNN	CONVOLUTIONAL NEURAL NETWORK
EAR	EYE ASPECT RATIO
FPS	FRAMES PER SECOND
HCI	HUMAN–COMPUTER INTERACTION
HOG	HISTOGRAM OF ORIENTED GRADIENTS
IMUTILS	IMAGE PROCESSING UTILITIES LIBRARY IN PYTHON
ML	MACHINE LEARNING
NUMPY	NUMERICAL PYTHON LIBRARY FOR ARRAY OPERATIONS
OPENCV	OPEN SOURCE COMPUTER VISION LIBRARY
PyAutoGUI	PYTHON AUTOMATION GRAPHICAL USER INTERFACE
ROI	REGION OF INTEREST
SVM	SUPPORT VECTOR MACHINE
UML	UNIFIED MODELING LANGUAGE
DFD	DATA FLOW DIAGRAM
RGB	RED GREEN BLUE (COLOR MODEL)
VGA	VIDEO GRAPHICS ARRAY (CAMERA RESOLUTION STANDARD)

Chapter 1

Introduction

1.1 BACKGROUND

Over the past few decades, computer technology has progressed from simple command-based systems to intelligent platforms that can interpret human intentions and behaviours. This evolution has led to the emergence of Human–Computer Interaction (HCI), a multidisciplinary field dedicated to designing communication methods that feel more natural and intuitive. Earlier, users depended on input devices such as keyboards, mice, or joysticks to operate digital systems. Although effective, these interfaces can be restrictive for people with physical limitations. The increasing emphasis on accessibility and convenience has encouraged the exploration of contact-free interaction techniques, enabling control through gestures, facial expressions, or eye movements instead of manual inputs.

Continuous advances in computer vision, machine learning, and image processing have made it possible to create systems that recognize and interpret facial cues in real time. By analyzing subtle changes in the eyes, mouth, and head position, computers can now respond to human gestures as input commands. Earlier approaches relied on costly infrared cameras or wearable sensors, which limited usability to specialized environments. Recent open-source tools such as OpenCV and Dlib, however, have removed this dependency by offering accurate facial landmark detection through an ordinary webcam. Building on these technological improvements, the present work develops a real-time facial-gesture-based mouse control system that promotes digital accessibility and inclusivity. It aims to provide a low-cost, software-driven alternative for users with mobility challenges while demonstrating the broader possibilities of vision-based interaction in everyday computing.

1.2 DOMAIN OF THE PROBLEM

Human–Computer Interaction (HCI) has become a key area of study focused on designing systems that allow people to communicate with computers in a more intuitive and natural way. While conventional input devices such as keyboards, mice, and touchscreens have served this purpose for many years, they are not suitable for everyone. Individuals who have physical impairments or limited motor control often struggle to operate these devices effectively. This limitation underscores the need for alternative, hands-free computing systems that interpret human gestures and facial expressions instead of relying solely on manual input. The central aim of this domain is to create technologies that can understand user intent through natural cues, enabling smooth and accessible interaction between humans and digital machines.

Among the many approaches explored to achieve this goal, vision-based interaction systems have shown great promise because they can analyze facial movements in real time using only a standard webcam. These systems apply computer-vision algorithms to identify and track features such as eye blinks, winks, and mouth movements, converting them into corresponding digital actions. The use of open-source frameworks such as OpenCV, Dlib, and PyAutoGUI makes it possible to build cost-effective and reliable systems without requiring any external sensors or specialized hardware. OpenCV handles video capture and image processing, Dlib detects facial landmarks accurately, and PyAutoGUI performs mouse operations based on the interpreted gestures. When combined, these tools allow for the creation of an efficient, affordable, and responsive vision-based interface suitable for a wide range of users.

Although a variety of assistive devices are available today, many depend on expensive infrared trackers or other specialized components, which limit their accessibility. The project presented in this report seeks to overcome those barriers by developing a software-driven facial-gesture mouse control system that functions entirely through a webcam. The system identifies facial features and interprets movement patterns to carry out actions such as cursor control, clicking, and scrolling. Beyond helping individuals with physical challenges, this approach is also valuable in environments where hands-free operation is essential such as medical facilities, laboratories, clean-room industries, and immersive applications like gaming or robotics. Positioned at the intersection of computer vision, machine learning, and assistive HCI, this domain emphasizes inclusivity, affordability, and adaptability. By converting simple facial gestures into digital commands, it demonstrates how vision-based technology can enhance accessibility and redefine the way humans interact with computers.

1.3 MOTIVATION

The motivation behind this project stems from the need to develop inclusive technologies that help individuals with physical disabilities interact with computers independently. Many people face challenges in using conventional input devices such as a mouse or keyboard due to limited motor control or physical constraints. Creating a vision-based system that interprets facial gestures as control inputs represents a step toward empowering these individuals and promoting digital inclusivity. By enabling interaction through natural expressions such as eye movement or mouth gestures, the project seeks to make computing more accessible and intuitive for all users.

Another major driving factor behind this project is the advancement of computer vision and machine learning, which now allow real-time interpretation of human facial cues with remarkable precision. Existing assistive systems often depend on expensive hardware, infrared sensors, or EEG-based devices that limit their widespread use. In contrast, this project utilizes only a standard webcam and open-source tools, making it an affordable and practical solution. The motivation lies in leveraging these technological innovations to bridge the accessibility gap and demonstrate that impactful solutions can be achieved through efficient, software-driven methods without the need for specialized hardware.

Beyond accessibility, this project is also inspired by the potential of hands-free interaction to transform the way humans engage with digital systems. In environments where manual operation is difficult or undesirable such as hospitals, laboratories, or industrial settings gesture-based control can enhance safety, convenience, and productivity. Therefore, the motivation extends beyond aiding individuals with disabilities; it reflects a broader vision of advancing Human–Computer Interaction toward a more natural, contact-free, and intelligent computing experience. This work embodies both a humanitarian goal and a technological innovation enhancing digital accessibility while redefining how people connect with machines.

1.4 PROBLEM STATEMENT

Conventional input devices such as keyboards and mice remain inaccessible or inefficient for individuals with motor impairments or limited physical mobility. Although several assistive technologies are available, many rely on costly sensors or complex hardware setups, making them impractical for widespread use. This gap highlights the need for a cost-effective and software-driven solution that allows hands-free control of computers through natural facial gestures.

The proposed project addresses this challenge by developing a vision-based facial gesture control system that utilizes Python, Dlib, and OpenCV to detect and interpret facial movements captured via a standard webcam. By recognizing gestures such as eye blinks, winks, and mouth openings, the system translates these actions into mouse operations including cursor movement, clicking, and scrolling. The approach eliminates the dependency on external devices, ensuring real-time performance, adaptability, and user comfort. Ultimately, this project seeks to bridge the accessibility gap in Human–Computer Interaction by enabling a reliable, scalable, and inclusive hands-free computing experience.

1.5 OBJECTIVES

1. To design and implement a real-time facial gesture recognition system capable of accurately detecting eye and mouth landmarks for gesture interpretation.

2. To integrate the gesture recognition module with PyAutoGUI to enable smooth mouse operations such as movement, left/right clicks, and scrolling.

3. To ensure system affordability and accessibility using only a standard webcam and open-source technologies.

4. To optimize detection performance by calibrating the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) thresholds for diverse lighting and facial conditions.

5. To contribute to inclusive technology by developing a practical, hands-free computing interface suitable for both assistive and general-purpose applications.

1.6 ADVANTAGES

- Provides complete hands-free control of the computer cursor.
- Enables users with physical disabilities to access computers more independently.
- Achieves real-time tracking and response through efficient facial feature recognition.
- Simulates standard mouse operations including single click, double click, and scroll using only facial gestures.
- Operates using a regular webcam without any external sensors or hardware attachments.

1.7 SCOPE OF THE PROJECT

The scope of this project focuses on designing and implementing a vision-based facial gesture control system that allows users to operate a computer mouse without using any physical input devices. The system uses a standard webcam to capture live video and applies computer vision algorithms to detect facial movements such as blinking, winking, and mouth opening. These gestures are processed through Python libraries like OpenCV, Dlib, and PyAutoGUI, which translate them into corresponding mouse operations such as cursor movement, clicking, and scrolling. The project aims to provide a cost-effective and accessible alternative for individuals with physical disabilities while also showcasing the broader potential of hands-free computing in everyday applications.

The system is entirely software-based and requires only basic hardware, making it portable, easy to use, and free from dependence on specialized sensors. It emphasizes real-time performance, detection accuracy, and user adaptability, achieved by fine-tuning parameters like the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) to ensure stable results under different conditions. While the current system focuses on essential mouse functionalities, it has the potential for future expansion by incorporating advanced gestures and machine learning models for more complex operations. Overall, the project contributes to the advancement of Human–Computer Interaction (HCI) by demonstrating how computer vision can enhance accessibility, inclusivity, and innovation in digital environments.

Chapter 2

Literature Survey

2.1 OVERVIEW OF HUMAN-COMPUTER INTERACTION

Human–Computer Interaction (HCI) is the study of how people communicate and interact with computer systems. It focuses on creating user-friendly designs that make technology easier and more efficient to use. Early computer systems required users to work with complex command-line interfaces and manual inputs through keyboards. Later, the invention of graphical interfaces and devices such as the mouse made interaction more intuitive. As technology progressed, the goal of HCI expanded toward creating natural and intelligent interaction methods that resemble real-world communication. The field now aims to make digital systems more adaptive, interactive, and accessible to users of all abilities.

However, traditional input devices like keyboards and mice are not suitable for everyone, especially individuals with physical disabilities or limited motor control. This has led researchers to explore contactless and hands-free interaction techniques such as speech recognition, gesture detection, and eye-tracking. Advances in computer vision, machine learning, and artificial intelligence have enabled systems to interpret facial expressions, eye blinks, and gaze direction in real time using only a standard webcam. These technologies are now used in various fields including assistive computing, gaming, healthcare, and automation. For this project, HCI serves as the foundation for developing a facial gesture-controlled mouse, allowing users to perform computer operations through simple eye and mouth movements. This approach not only improves accessibility but also promotes a more natural and inclusive way of interacting with technology.

2.2 OVERVIEW OF EYE-TRACKING TECHNOLOGIES

Eye-tracking technology is a method used to monitor and analyze the movement of a person's eyes to understand where they are looking and how their gaze changes over time. This technology can detect actions such as blinking, winking, and gaze direction, which can then be used as input commands for computers or other digital systems. Early eye-tracking systems relied on infrared sensors and specialized cameras to capture detailed eye movements, providing high accuracy but at a very high cost. These systems were mainly used for research, medical studies, and advanced human–computer interaction experiments. With the rapid growth of computer vision and image processing, modern eye-tracking systems can now work effectively using a simple webcam, reducing the need for expensive hardware while maintaining acceptable accuracy and speed.

Today, eye-tracking plays an important role in many applications such as assistive technologies, gaming, psychology, marketing, and virtual reality. In assistive computing, it allows individuals with physical disabilities to control a computer or cursor using only their eye movements. For example, blinking can be used to simulate mouse clicks, while gaze direction can move the cursor. Open-source software tools and libraries like OpenCV and Dlib have made it easier to design such systems by providing reliable face and eye detection algorithms. These advancements have helped transform eye-tracking from a complex laboratory setup into a practical, low-cost solution that can be implemented on regular computers. Overall, the evolution of eye-tracking technologies has made hands-free and gaze-controlled interaction more accessible, opening the door for inclusive and user-friendly computing.

2.3 OVERVIEW OF FACIAL GESTURE RECOGNITION SYSTEMS

Facial gesture recognition is a technology that allows computers to understand human expressions and facial movements. It works by detecting specific points on the face, known as facial landmarks, and tracking how these points move when a person blinks, smiles, or opens their mouth. Using these movements, the system can interpret different gestures and convert them into commands that a computer can understand. Earlier, such systems required special sensors or cameras to detect depth and facial features accurately. However, with the development of advanced computer vision techniques, real-time facial gesture recognition can now be achieved using ordinary webcams and open-source libraries. This makes the technology more accessible and affordable while maintaining good accuracy for tasks such as cursor control, emotion detection, and user authentication.

Modern facial gesture recognition relies heavily on algorithms that can identify and track facial landmarks efficiently. Libraries like Dlib and OpenCV are commonly used to detect and analyze key points around the eyes, mouth, and nose. Metrics such as the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) are used to measure blinking and mouth opening, allowing the system to distinguish between different facial gestures. These technologies have a wide range of applications, including hands-free computing, gaming, healthcare, and human–robot interaction. In this project, facial gesture recognition is used to control mouse operations such as moving the cursor, clicking, and scrolling through simple facial actions. This approach helps promote accessibility and demonstrates how human expressions can serve as an effective, natural means of computer interaction.

2.4 REVIEW OF RELATED WORK

In recent years, the advancement of computer vision and machine learning has led to remarkable progress in human–computer interaction (HCI), particularly in facial and eye-based control systems. Numerous researchers have explored real-time, affordable, and user-friendly alternatives to traditional input devices, such as the mouse and keyboard. This section presents a review of significant research contributions in the areas of eye-tracking, facial gesture recognition, and multimodal interaction systems that have advanced the field of vision-based assistive technologies.

2.4.1 CLASSICAL AND GEOMETRIC-BASED APPROACHES

Several researchers have explored classical geometric and rule-based frameworks to interpret eye and facial gestures for computer control. Mygapula et al. [1] developed an iris-based computer control system that translates eye movements into cursor motion. The model employed an iris detection mechanism using OpenCV and Python, which mapped gaze directions to screen coordinates, and blink gestures were used for mouse-click operations. The system achieved more than 90% accuracy even under fluctuating illumination, demonstrating the feasibility of cost-effective webcam-based assistive solutions for physically challenged users.

Vasisht et al. [10] introduced an eye-controlled HCI system that combined eye and facial feature tracking for precise cursor manipulation. The design utilized facial landmarks derived from the Dlib library, where eye blinking and mouth movements were interpreted as click and control actions, respectively. The framework leveraged the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) to detect blinking and mouth opening in real time, ensuring smoother and more accurate performance using standard webcams without additional hardware.

Sreeni et al. [8] proposed a vision-based cursor control system driven by facial movements. Using Haar Cascade and Adaboost algorithms, the system detected head motion and facial positions, translating them into directional cursor movements. Eye blinking was interpreted as click events, and facial orientation guided pointer direction. Their approach effectively enabled disabled individuals to perform essential computer operations such as file navigation and selection through simple facial gestures.

These geometric and classical methods established a reliable foundation for webcam-based real-time control systems. However, they exhibited limitations under inconsistent lighting, varied face orientations, and background complexity. These challenges encouraged further advancements through hybrid and deep learning-driven models for improved accuracy.

2.4.2 HYBRID AND DEEP LEARNING–BASED APPROACHES

To overcome the constraints of purely geometric systems, researchers began integrating traditional image processing with deep learning for greater adaptability. Saradha et al. [2] developed a hybrid gaze-controlled cursor system using Histogram of Oriented Gradients (HOG) for facial detection, followed by a Convolutional Neural Network (CNN) for classifying gaze direction and eye states. This hybrid approach improved the model's resilience to lighting variations and facial orientation changes, achieving high precision with low computational cost suitable for real-time applications.

Kanakaprabha et al. [6] proposed a regression-assisted hybrid framework that utilized the Hough Transform for iris localization coupled with regression analysis to predict gaze coordinates. This technique improved tracking accuracy while reducing latency, demonstrating efficient performance in real-time environments without reliance on high-end GPUs.

Miah et al. [7] introduced an innovative deep learning–driven approach using Mediapipe and EAR-based blink detection. Their system employed convolutional modeling to interpret gaze directions and blink gestures, combining feature-based and appearance-based eye-tracking methods. The framework enhanced stability through adaptive calibration and demonstrated high usability for disabled users requiring non-contact interaction systems.

Collectively, these studies illustrate the benefits of hybridized computer vision models that merge classical detection with neural feature extraction. They deliver a balanced trade-off between real-time speed, accuracy, and computational efficiency, marking a crucial evolution toward intelligent and adaptive eye-tracking systems.

2.4.3 MULTIMODAL AND MEDIAPIPE-BASED SYSTEMS

Recent advancements in HCI have emphasized multimodal control systems that combine multiple sensory inputs such as hand, face, and eye gestures to provide flexible and intuitive interaction. Jayalakshmi et al. [9] designed a multimodal virtual mouse system that incorporated both hand and eye gestures using OpenCV, Mediapipe and PyAutoGUI. The interface allowed users to perform diverse tasks such as cursor navigation, volume control, and clicking through synchronized visual inputs. This system demonstrated that fusing visual modalities can enhance precision, reduce fatigue, and support inclusive computing for users with mobility impairments.

Murthy et al. [5] presented an eye gesture–based virtual mouse using a webcam and Python for pointer control. The model employed Haar cascade classifiers for face detection and utilized gaze and pupil tracking to enable smooth and natural cursor movement. The design emphasized accessibility by avoiding specialized sensors, showcasing that standard cameras can support high-accuracy gaze-based systems for assistive purposes.

Similarly, Abiyev and Arslan [4] developed a CNN-driven head and eye control system for individuals with motor disabilities. The system employed deep neural feature extraction to interpret facial and head gestures, enabling users to navigate and click on-screen elements efficiently. Their model achieved high stability and responsiveness, validating CNN's capability to enhance precision in human–computer communication.

Collectively, multimodal and Mediapipe powered frameworks signify a new phase of advancement in visual interaction technologies. They offer superior adaptability, robustness, and multi-functionality, though challenges remain in user calibration, illumination adaptation, and long-term tracking reliability. Nonetheless, these innovations pave the way for more inclusive, context-aware, and intelligent assistive systems.

2.4.4 SUMMARY OF REVIEWED WORKS

The reviewed studies demonstrate a progressive transition in vision-based human–computer interaction—from basic geometric tracking to advanced hybrid and multimodal systems. Early research validated the practicality of webcam-based control through facial landmarks and aspect ratio computations, while more recent works adopted CNNs, regression models, and dense landmark extraction for improved robustness and real-time adaptability.

Despite the rapid improvements, several issues persist, including sensitivity to lighting changes, high computational requirements, and limited scalability across different users and environments. Many existing systems still depend on high-performance hardware or extensive calibration, hindering universal accessibility.

The proposed project addresses these shortcomings by introducing a purely software-driven real-time facial gesture control system using Python, OpenCV, and Dlib. By leveraging natural gestures such as blinking, winking, and mouth movements, the system provides a hands-free, low-cost, and efficient interface, making computing more inclusive and accessible to all users.

s

Chapter 3

System Specification

The Eye-Controlled Mouse Using Facial Gestures is a vision-based system designed to perform mouse operations through facial movements, providing an effective hands-free computer interaction method. The system captures real-timeanalyzedrom a webcam and processes each frame to detect facial landmarks such as eyes, mouth, and nose using computer vision algorithms. These landmarks are then analyzed to identify gestures like blinking, winking, or mouth opening, which are translated into mouse commands such as cursor movement, clicking, or scrolling

3.1 HARDWARE REQUIREMENTS

The proposed system is designed to run on standard computing hardware and does not require any special external sensors or devices. Since it processes real-time video data, a moderate level of computational performance and an HD webcam are sufficient for smooth operation.

Minimum Configuration:

Processor	: Intel Core i3 or equivalent (minimum 2.0 GHz)
RAM	: 4 GB or higher
Storage	: 500 GB HDD or 128 GB SSD
Webcam	: Standard HD (720p) or higher
Display	: 1024×768 resolution or above
Input/Output Devices	: Standard keyboard and monitor (for initial setup and testing)

Recommended Configuration:

Processor	: Intel Core i5/i7 or AMD Ryzen 5 or higher
RAM	: 8 GB or higher
Webcam	: Full HD (1080p) supporting 30 FPS or above
Graphics Support	: Optional GPU for faster frame processing

The overall responsiveness of the system depends on the processing power and the webcam's frame rate. Higher-quality cameras and better lighting conditions enhance detection precision and reduce frame delays during gesture tracking.

3.2 SOFTWARE REQUIREMENTS

The system is implemented using Python, which provides robust support for image processing and automation through its open-source libraries. The combination of libraries such as OpenCV, Dlib, and PyAutoGUI allows efficient face detection, gesture analysis, and simulated mouse operations.

Software Configuration:

Operating System	: Windows 10/11 or Ubuntu Linux
Programming Language	: Python (version 3.8 or higher)
Libraries Used	: OpenCV, Dlib, NumPy, Imutils, PyAutoGUI
Model File	: shape_predictor_68_face_landmarks.dat (Dlib pre-trained)

Chapter 4

System Design

4.1 SYSTEM OVERVIEW:

The proposed system is designed to enable hands-free computer interaction using facial gestures detected through a standard webcam. Instead of relying on physical devices such as a mouse or keyboard, this system interprets facial movements to perform equivalent mouse operations in real time. The goal is to assist individuals with limited motor control while also offering a new mode of interaction for general users.

The system operates based on the following key facial actions:

- Squinting the eyes
- Winking (one eye closure)
- Head movement (pitch and yaw orientation)
- Opening the mouth

Each of these gestures corresponds to a specific mouse function, such as movement, clicking, or scrolling. By combining face detection, landmark localization, and gesture interpretation, the system delivers smooth, real-time control of the cursor without external sensors.

4.2 SYSTEM ARCHITECTURE

The system architecture represents the structural design and data flow between various modules of the Eye-Controlled Mouse System. It describes how input from the webcam is processed through several layers such as face detection, feature extraction, and gesture interpretation to produce the desired output actions, like mouse movement or clicking.

The design follows a modular approach, ensuring that each component performs a specific task and interacts seamlessly with others. The architecture enhances efficiency, scalability, and ease of debugging during implementation.

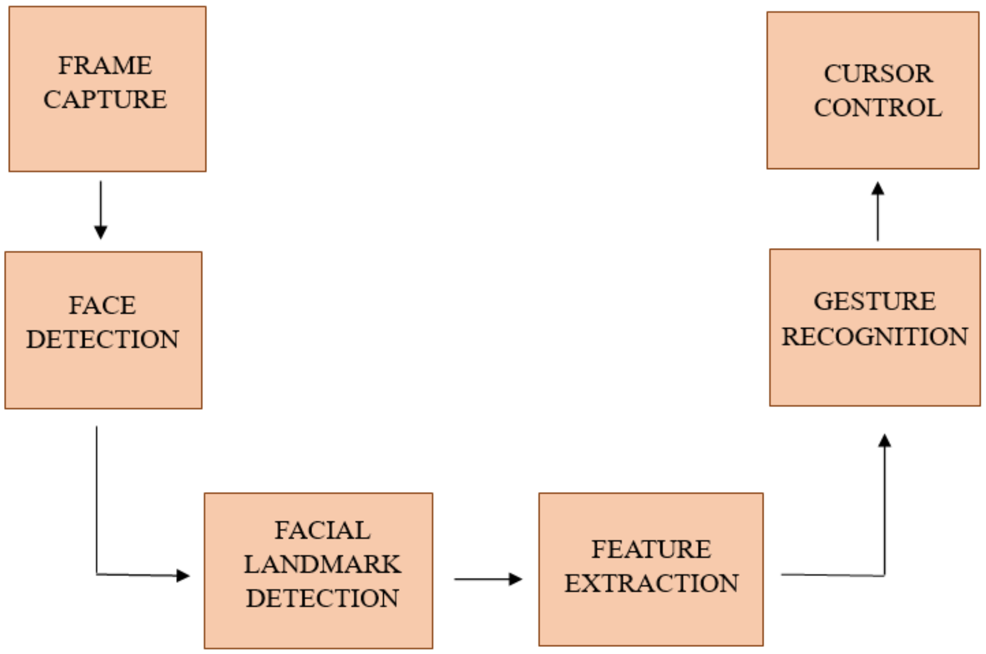

Figure 4.1: System Architecture

4.3 MODULE DESCRIPTION

4.3.1 FRAME CAPTURE

This module activates the system's webcam to continuously capture live video input from the user. Each frame is processed at a rate of around 30 frames per second (FPS), providing a stable and responsive stream. These frames act as the foundation for further processing, including face detection, feature extraction, and gesture recognition. Consistent frame acquisition ensures uninterrupted real-time interaction.

4.3.2 FACE DETECTION

The face detection module is responsible for identifying the user's facial region from each captured frame. It uses Dlib's HOG (Histogram of Oriented Gradients) based frontal face detector to locate the face accurately. Once detected, a bounding box is drawn around the facial area, which is then passed to the next module for landmark detection. By isolating only the relevant region of interest, this module ensures precise and efficient facial analysis.

4.3.3 FACIAL LANDMARK DETECTION

After the face has been detected, this module determines the key feature points on the face using Dlib's pre-trained model. It maps 68 distinct landmark points corresponding to essential facial regions such as the eyes, nose, and mouth. These landmark coordinates are extracted and stored in a structured format for further geometric calculations. This step provides the necessary data foundation for identifying subtle facial movements.

4.3.4 FEATURE EXTRACTION (EYES AND MOUTH)

In this stage, the eyes and mouth are isolated from the detected landmarks for focused analysis. Two important parameters are calculated. Eye Aspect Ratio (EAR) and the Mouth Aspect Ratio (MAR) using the distances between predefined landmark points. These ratios help determine whether the eyes or mouth are open or closed. This information is then utilized to recognize various facial gestures that will later control computer functions.

4.3.5 GESTURE RECOGNITION

This module analyzes the EAR and MAR values obtained from the feature extraction stage and classifies them into specific gestures. Examples include:

- Both eyes closed → Scroll or double-click
- Left eye closed → Left-click
- Right eye closed → Right-click
- Mouth open → Activate or deactivate cursor control

Predefined threshold limits help distinguish between regular facial movements and deliberate gestures. The accurate recognition of these gestures enables reliable hands-free operation.

4.3.6 CURSOR CONTROL

The final module translates the recognized gestures into corresponding computer mouse actions using the PyAutoGUI library. Cursor movement is controlled by tracking head orientation, while detected gestures trigger mouse events such as clicks or scrolling. The module ensures minimal delay between gesture detection and command execution, allowing smooth, real-time interaction that mirrors traditional mouse functionality.

4.4 UML DIAGRAMS

4.4.1 USE CASE DIAGRAM

A Use Case Diagram in the Unified Modeling Language (UML) is a behavioral diagram developed through use-case analysis. It provides a visual representation of a system's functional behavior by illustrating the relationship between actors (users or external entities) and the use cases (system functions) they interact with. The diagram highlights how different users engage with the system and what specific operations are available to them. In essence, it defines who does what within the system and helps in understanding the system's overall functionality and user interactions at a high level.

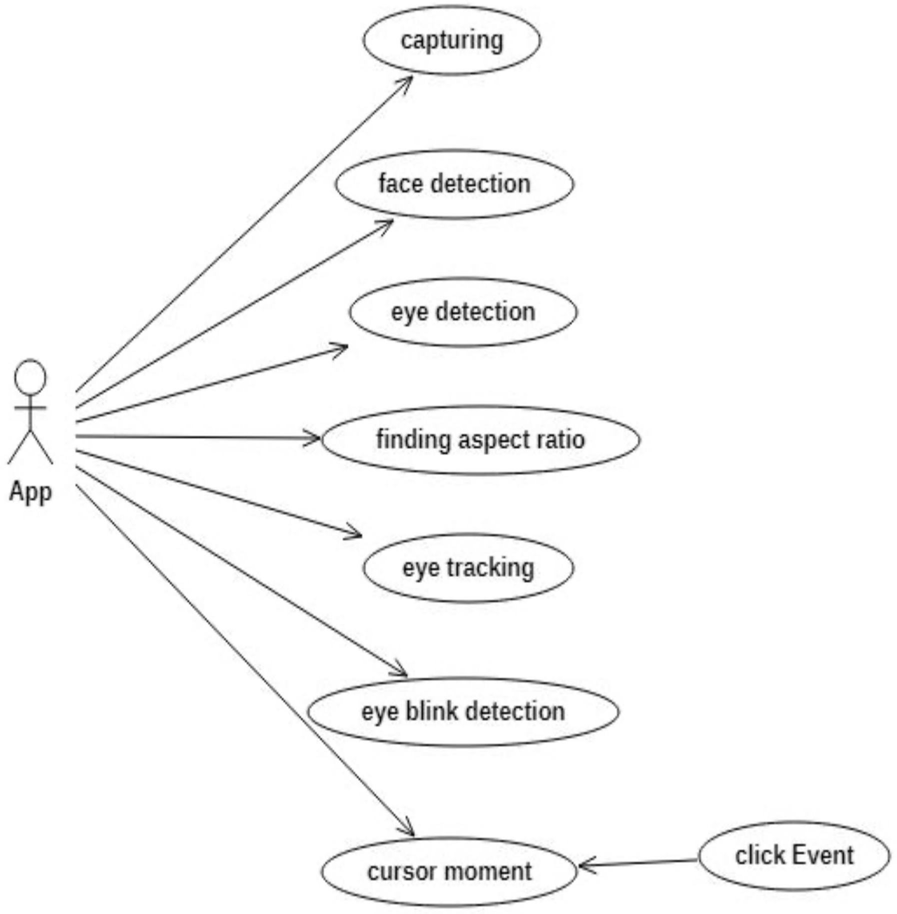

Figure 4.2: Use Case Diagram

4.4.2 CLASS DIAGRAM

Class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

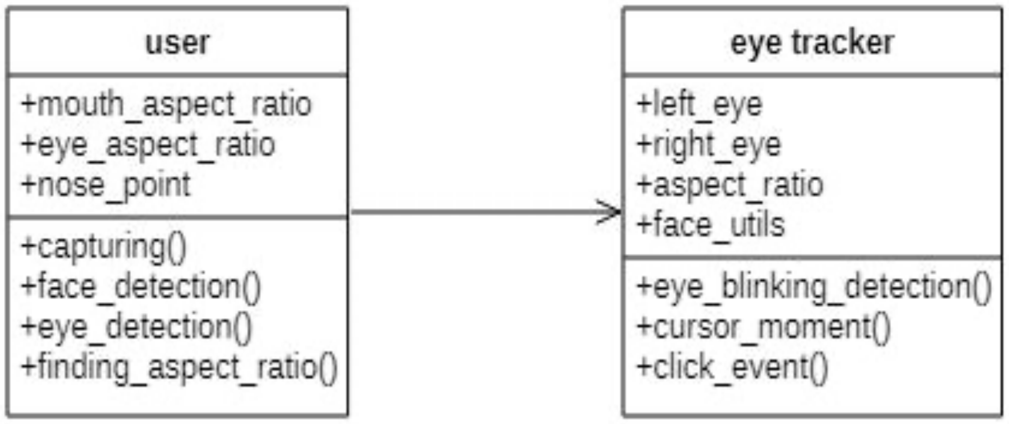

Figure 4.3: Class Diagram Representing Module Relationships

4.4.3 SEQUENCE DIAGRAM

Sequence Diagrams represents how different objects interact over time, where the objects are shown horizontally, and the progression of time is depicted vertically. It illustrates the flow of messages between objects to achieve a particular function or process. A Use Case, on the other hand, is a type of behavioral model that describes the actions a system can perform in collaboration with one or more external actors. Each use case outlines a specific functionality or service offered by the system, without detailing its internal design. These interactions between the system and the actor can lead to changes in the system's state or trigger communications with external components. Additionally, a use case may include variations such as alternate flows, exceptions, or error-handling scenarios to represent different possible outcomes of the interaction.

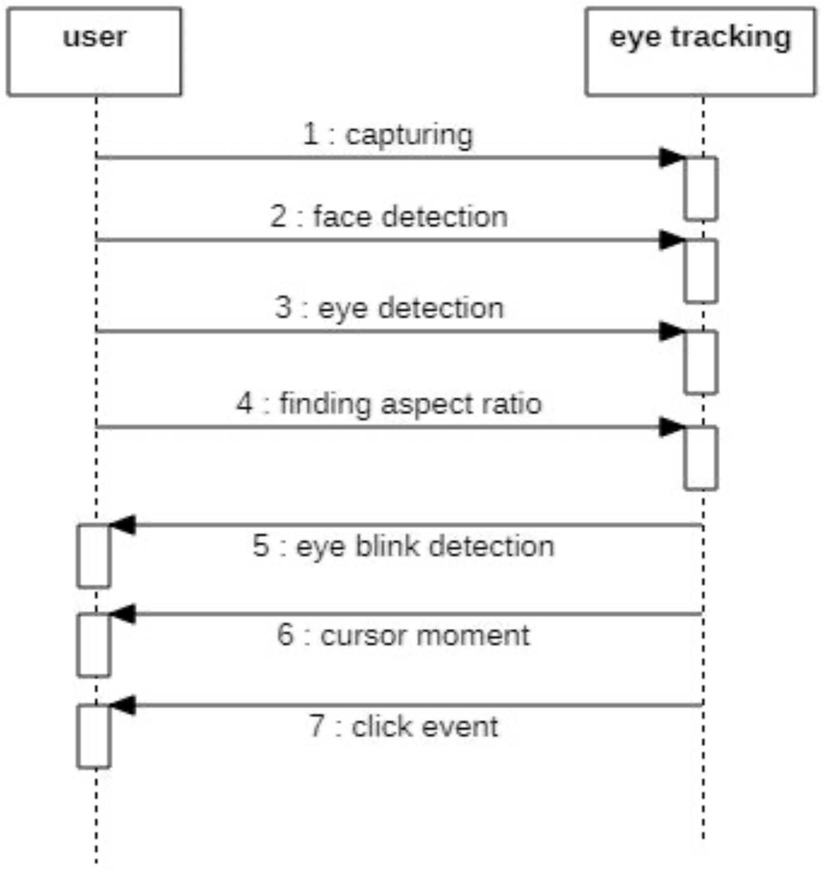

Figure 4.4: Sequence Diagram

4.4.4 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

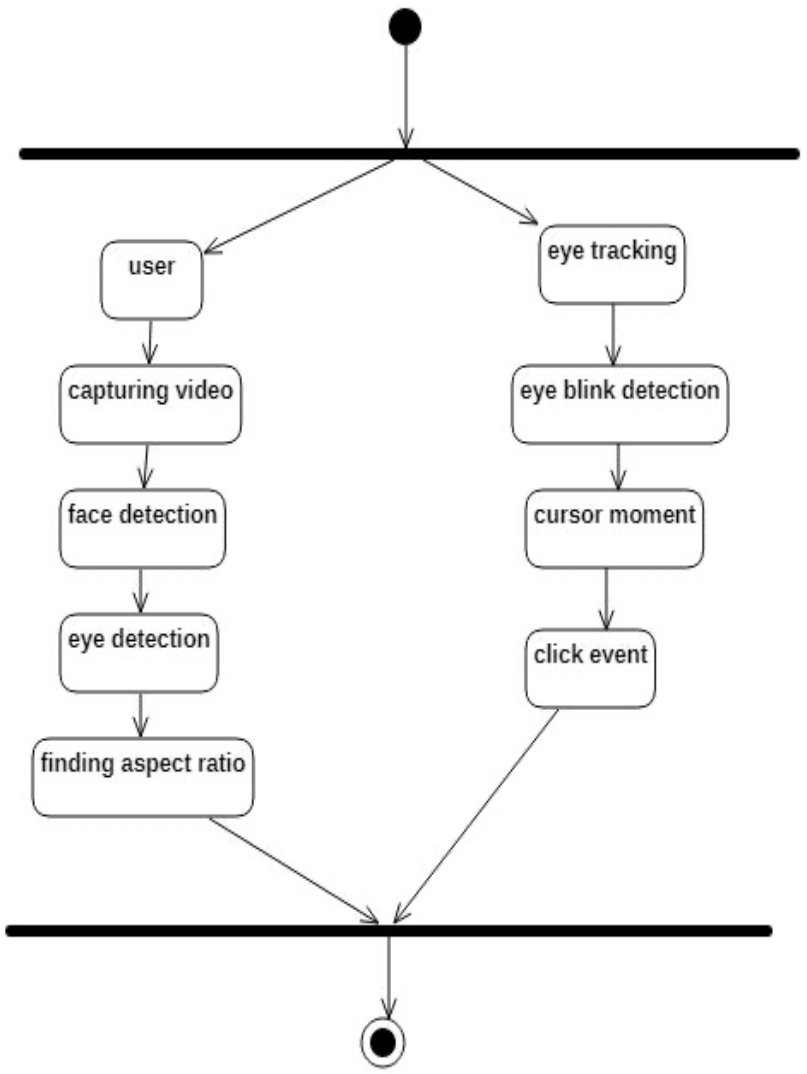

Figure 4.5: Activity Diagram

4.4.5 STATE CHART DIAGRAM

A Statechart Diagram is one of the key UML diagrams used to represent the dynamic behavior of a system. It illustrates the various states an object can occupy during its lifetime and the transitions between these states triggered by specific events. Such diagrams are particularly useful for modeling reactive systems, which respond to internal or external events.

The main objective of a Statechart Diagram is to depict how an object's state changes from creation to termination based on certain actions or conditions. It helps in understanding the control flow and behavior of an object over time. These diagrams are also valuable in both forward and reverse engineering processes, providing a clear view of system dynamics.

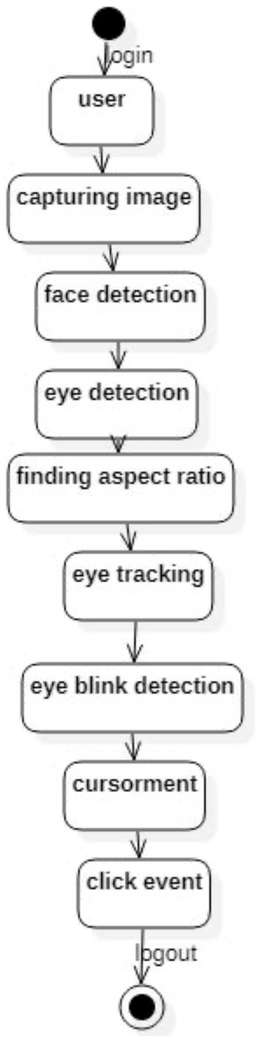

Figure 4.6: State Chart Diagram

4.5 DATA FLOW

The Data Flow Diagram (DFD), also known as a bubble chart, is a graphical representation that illustrates the flow of data within a system. It shows how input information is processed through various stages to produce the desired output. The DFD provides a clear visual understanding of how data moves through the proposed system and how each module interacts with one another to perform specific operations.

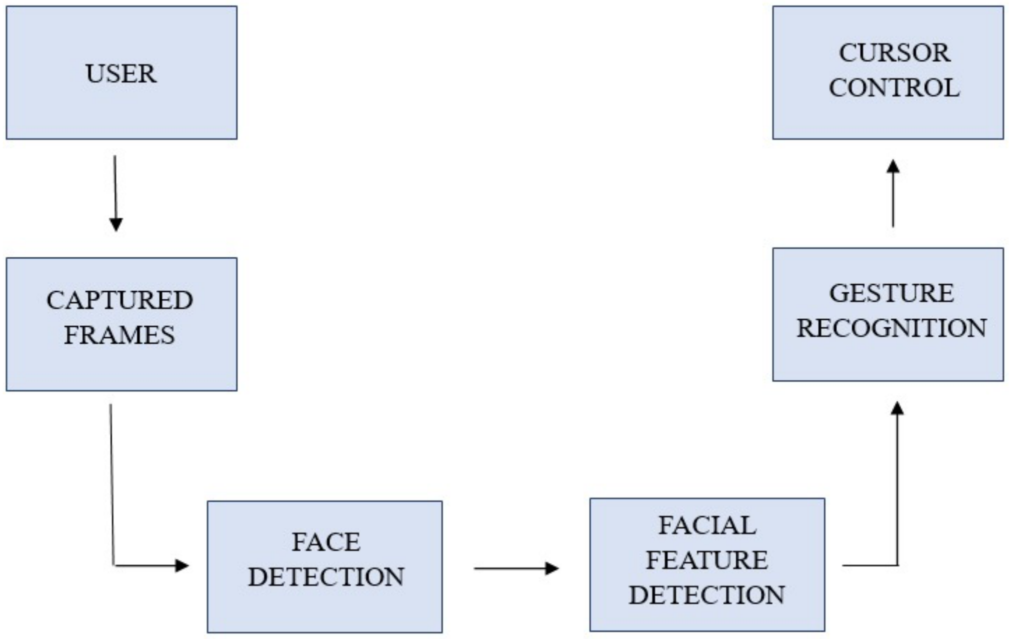

Figure 4.7: Data Flow Diagram of the System

In this project, the DFD represents the data flow of the Eye-Controlled Mouse System, which performs computer control using facial gestures. The flow begins with the activation of the webcam to capture live video frames. Each frame undergoes preprocessing steps such as resizing, flipping, and conversion to grayscale before being passed to the detection stage. The system then identifies the face using Dlib's HOG-based detector and extracts facial landmarks such as the eyes, nose, and mouth.

After facial landmarks are extracted, the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) values are computed to identify specific gestures. The recognized gestures, including blinking, winking, and mouth opening, are mapped to corresponding mouse operations such as clicks, scrolling, or cursor movement.

This sequential data flow ensures smooth and efficient real-time performance, allowing the user to interact with the computer naturally and hands-free.

Chapter 5

Methodology

5.1 METHODOLOGY

The proposed methodology outlines the structured approach followed in developing the Eye-Controlled Mouse System Using Facial Landmarks and Gesture Recognition. The system leverages computer vision techniques to detect facial gestures and translate them into corresponding mouse operations, allowing hands-free interaction with a computer.

Initially, the webcam is activated to capture real-time video input at around 30 frames per second. Each captured frame undergoes basic preprocessing such as resizing, flipping, and conversion to grayscale to improve detection accuracy and reduce computational load. The processed frame is then analyzed using Dlib's pre-trained facial landmark predictor, which identifies key facial features including the eyes, mouth, and nose.

From these detected features, two geometric ratios Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) are computed to recognize user gestures like blinking, winking, and mouth opening. These gestures are then mapped to mouse operations such as cursor movement, clicking, and scrolling using the PyAutoGUI library.

This sequence of operations is executed continuously in real time, ensuring smooth and responsive interaction between the user's facial gestures and system actions.

5.2 FUNCTIONAL MODULES

The proposed system comprises several interdependent modules that work together to perform image acquisition, facial feature extraction, gesture recognition, and cursor control. Each module ensures high detection accuracy, real-time performance, and smooth interaction.

5.2.1 RESIZING

The webcam captures live video frames at around 30 frames per second. Each frame is resized to a standard resolution (e.g., 640×480 pixels) to maintain consistency and reduce computational load. The frame is horizontally flipped to match natural mirror-like movement, ensuring intuitive control when the user moves right, the cursor moves right.

5.2.2 CONVERSION FROM BGR TO GRAY

OpenCV captures images in the BGR (Blue-Green-Red) format, which contains three color channels. Since color information is unnecessary for facial landmark detection, each frame is converted to grayscale. This conversion simplifies the data, reduces memory consumption, and accelerates detection without compromising precision.

5.2.3 DETECTION AND PREDICTION OF FACIAL FEATURES

Facial detection and prediction are carried out using Dlib's pre-trained model, which identifies the presence and location of a face within each captured frame. A built-in function known as detector() is utilized by the model to detect the face in the input image. Once the facial region has been identified, another function called predictor() is employed to determine the specific facial landmarks.

Through this function, 68 landmark points are mapped onto the two-dimensional facial image. Each of these points corresponds to a unique region on the face, such as the eyes, nose, and mouth. The coordinates generated by the function are represented in a two-dimensional (x, y) format, where each pair collectively defines a specific point on the facial structure. When these points are connected sequentially, they outline the approximate geometry of the human face.

The extracted coordinates are stored in an array structure so that they can be processed and utilized in subsequent operations. From this array, four distinct subsets of coordinate values are created, representing the left eye, right eye, nose, and mouth regions. These subsets are maintained separately to facilitate precise region-based analysis. After the subsets are prepared, contours are drawn around the eyes and mouth by connecting the respective landmark points. This process is executed using the drawContour() function, which generates boundary outlines around the defined regions. The resulting contours visually represent the detected facial areas and serve as the foundation for subsequent gesture-based actions.

5.2.4 MOUTH AND EYE ASPECT RATIOS

Once the contours are drawn, it is necessary to have a reference for the shapes, which, when compared with, gives the program any information about any action made by these regions such as blinking, yawing, etc. These references are understood as ratios, between the 2D coordinates, and a change in the coordinates, essentially tell us that, the parts of the region of the face have moved from the regular position and an action has been performed. The system is built on predicting facial landmarks of the face. The Dlib prebuilt model helps in fast and accurate face detection along with 68 2D facial landmarks as explained already. Here, Eye-AspectRatio (EAR) and mouth-aspect-ratio (MAR) are used to detect blinking/winking and yawing respectively. These actions are translated into mouse actions.

Similarly, the MAR goes up when the mouth opens. This is used as an action to start and switch off the mouse. For instance, if the ratio has increased, it can mean that the distances between the points representing the region of the face have changed and an action has been performed by the person. This action is supposed to be understood as the person trying to perform an operation using the mouse. Hence, for these functionalities to be made operational, there need to be some defined ‘aspect_ratios’, which when cross a defined limit, interprets an action being performed.

5.2.5 ACTION DETECTION

After the ratios are defined, the frame can now compare the ratios of the parts of the face with the ratios defined for different actions, of the current frame being processed. It is done using the ‘if’ statement. The actions which the program identifies are:

1. For activating the mouse: The user needs to ‘yaw’ which is opening his mouth, vertically, in turn increasing the distance between the corresponding 2D points of the mouth. The algorithm detects the change in the distance by computing the ratio, and when this ratio crosses a specified threshold, the system is activated and the cursor can be moved. The user needs to place his nose towards, either the top, bottom, left or right of a rectangle that appears, to move the cursor in the corresponding direction. The more he is away from the rectangle, the faster is the movement of the cursor

2. Left/Right Clicking: For clicking, he needs to close any one of his eye, and make sure to keep the other open. The program first checks whether the magnitude of the difference is greater than the prescribed threshold by using the difference between the ratios of the two eyes, to make sure that the user wants to perform either the left or right click, and does not want to scroll(For which both the eyes need to squint)

3. Scrolling: The user can scroll the mouse, either upwards or downwards. The user needs to squint his eyes in such a way that the aspect ratio of both the eyes is less than the prescribed value. In this case, when the user places his nose outside the rectangle, the mouse performs scroll function, rather than moving the cursor. The user can move his nose either above the rectangle to scroll upwards, or move it below the rectangle to scroll downwards.

5.3 ALGORITHM AND FLOW CHART

5.3.1 ALGORITHM OF THE PROPOSED SYSTEM

The proposed system follows a structured sequence of steps for detecting facial gestures and translating them into mouse operations using OpenCV and Dlib. The following algorithm outlines the workflow logic:

1. Start the system and initialize all required libraries such as OpenCV, Dlib, Imutils, NumPy, and PyAutoGUI.

2. Load the pre-trained facial landmark model(shape_predictor_68_face_landmarks.dat) for detecting facial key points.

3. Activate the webcam to continuously capture live video frames at a rate of approximately 30 frames per second.

4. Read each frame and perform the following preprocessing steps:
   - Flip the frame horizontally to maintain a mirror-like orientation.
   - Resize the frame to a fixed resolution (e.g., 640×480).
   - Convert the color image (BGR format) into grayscale for faster and more efficient facial landmark detection.

5. Detect the face region in the grayscale frame using Dlib's frontal face detector (HOG-based).

6. Predict facial landmarks for the detected face and extract specific regions of interest: eyes, mouth, and nose.

7. Compute feature-based metrics:
   - Eye Aspect Ratio (EAR): measures the distance between eyelids to detect blinks or winks.
   - Mouth Aspect Ratio (MAR): measures mouth opening to identify mouth-based gestures.

8. Apply decision thresholds for each aspect ratio to determine gesture types:
   - Both eyes closed → Scroll Mode
   - Left eye wink → Left Click
   - Right eye wink → Right Click
   - Mouth open → Enable/Disable Cursor Movement Mode

9. Track the nose position to determine cursor direction (up, down, left, right) and move the cursor accordingly.

10. Perform corresponding mouse actions through the PyAutoGUI library based on recognized gestures.

11. Continuously repeat the process for each new frame.

12. Stop the webcam stream and terminate the program gracefully.

5.3.2 FLOW CHART

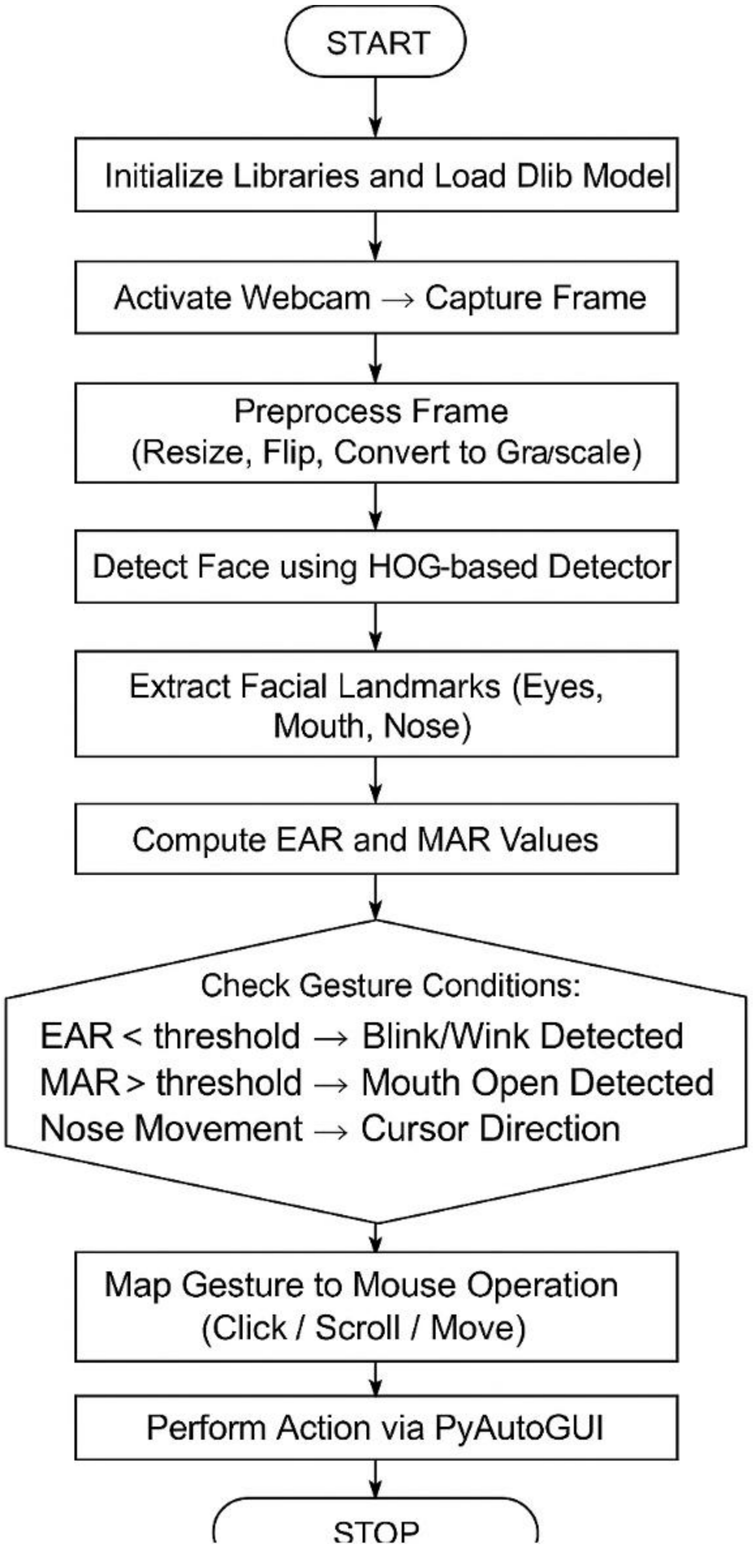
START
Initialize Libraries and Load Dlib Model
Activate Webcam → Capture Frame
Preprocess Frame (Resize, Flip, Convert to Grayscale)
Detect Face using HOG-based Detector
Extract Facial Landmarks (Eyes, Mouth, Nose)
Compute EAR and MAR Values
Check Gesture Conditions:
EAR < threshold → Blink/Wink Detected
MAR > threshold → Mouth Open Detected
Nose Movement → Cursor Direction
Map Gesture to Mouse Operation (Click / Scroll / Move)
Perform Action via PyAutoGUI
STOP

Figure 5.1: Flowchart of Eye and Facial Feature Detection Process

Chapter 6

Implementation

6.1 TOOLS USED

6.1.1 PYTHON

Python is an interpreter-based, object-oriented, high-level programming language with dynamic semantics. It is known for its simplicity, readability, and flexibility, which makes it suitable for beginners as well as professionals. Python's rich set of built-in data structures, along with dynamic typing and binding, make it highly efficient for rapid application development. It also supports scripting and integration with other languages, allowing it to serve as a "glue language" for connecting various software components.

In this project, Python acted as the primary development language for implementing facial landmark detection and real-time eye gesture recognition. Its extensive libraries such as OpenCV, Dlib, NumPy, and PyAutoGUI enabled easy integration of image processing, mathematical computation, and hardware control functionalities. Python's modular nature and support for packages encouraged code reusability and modular design, making the project more structured and manageable.

Additionally, Python's interactive development environment allows faster prototyping and debugging, as there is no need for a compilation step. Errors are handled gracefully through exceptions, and tools such as the built-in debugger simplify error tracing. The availability of Python on all major operating systems and its free, open-source nature further contribute to its popularity as a versatile tool for academic research and real-time application development.

6.1.2 ANACONDA

Anaconda is an open-source distribution platform for Python and R designed specifically for data science, artificial intelligence, and machine learning applications. It simplifies package management, environment setup, and dependency handling, allowing developers to focus more on implementation rather than configuration issues. Anaconda provides a robust ecosystem containing over 300 pre-installed libraries for numerical computation, data analysis, and visualization, making it a preferred environment for researchers and developers.

In this project, Anaconda was used as the core development environment to execute the Python scripts and manage required packages such as OpenCV, Dlib, imutils, and NumPy. Its integrated Conda package manager ensured smooth installation of dependencies and avoided version conflicts that commonly occur in traditional setups. The Anaconda Navigator interface provided an easy-to-use graphical interface to launch tools like Jupyter Notebook and Spyder, which enhanced the process of code writing, testing, and visualization.

Anaconda also provides excellent reproducibility and environment isolation, which are crucial for academic projects. By creating a virtual environment, the project could be executed on any system without affecting global settings or other applications. Its combination of stability, scalability, and user-friendly tools makes Anaconda one of the best platforms for executing Python-based computer vision projects like the Eye-Controlled Mouse system.

6.2 LIBRARIES USED

6.2.1 OPEN CV

OpenCV, short for Open Source Computer Vision Library, is one of the most widely used computer vision frameworks in Python. It provides a vast collection of functions for real-time image and video processing. In this project, OpenCV was used to capture frames from the webcam, convert them into grayscale images, and visualize the detected facial features on the video stream. The library also supported geometric transformations such as resizing, flipping, and drawing contours around facial landmarks. Its ability to process images at high speed made it ideal for implementing real-time gesture-based control.

6.2.2 DLIB

Dlib is a powerful machine learning and computer vision toolkit that provides robust face detection and facial landmark estimation models. The project utilized Dlib's pre-trained model, shape_predictor_68_face_landmarks.dat, to detect 68 distinct points on a human face, including the eyes, mouth, nose, and jawline. These landmarks were essential for calculating aspect ratios to determine eye blinks, mouth openings, and nose movements. Dlib's precision and reliability made it a key component in achieving accurate gesture recognition.

6.2.3 IMUTILS

The imutils library is a lightweight Python package that simplifies common image processing tasks such as resizing, rotation, and landmark indexing. It was used in this project to streamline operations like adjusting frame dimensions and accessing specific facial landmark indices for the eyes, mouth, and nose. By using imutils, the overall code became more readable and concise, reducing the complexity of manual computations often required with OpenCV.

6.2.4 NUMPY

NumPy, short for Numerical Python, is a fundamental library for numerical computing in Python. It provides efficient data structures and mathematical functions for handling arrays and matrices. In this project, NumPy was used to perform mathematical operations such as calculating Euclidean distances, averaging coordinates, and deriving aspect ratios for the eyes and mouth. Its ability to handle numerical data efficiently contributed to faster computations and smoother frame analysis.

6.2.5 PYAUTOGUI

PyAutoGUI is a Python module that enables automation of keyboard and mouse functions. It was used in this system to translate detected facial gestures into corresponding mouse actions such as movement, clicking, and scrolling. For example, winking triggered a left or right click, while nose movement controlled the cursor direction. PyAutoGUI served as the bridge between computer vision input and hardware control, allowing for seamless, hands-free computer interaction.

6.3 CODE

```
from imutils import face_utils
from utils import *
import numpy as np
import pyautogui as pag
import imutils
import dlib
import cv2

from threading import Thread

import pyttsx3
engine=pyttsx3.init('sapi5')
voices=engine.getProperty('voices')
engine.setProperty('voice','voices[0].id')

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

# Thresholds and consecutive frame length for triggering the mouse action.
MOUTH_AR_THRESH = 0.6
MOUTH_AR_CONSECUTIVE_FRAMES = 15
EYE_AR_THRESH = 0.19
EYE_AR_CONSECUTIVE_FRAMES = 15
WINK_AR_DIFF_THRESH = 0.04
WINK_AR_CLOSE_THRESH = 0.19
WINK_CONSECUTIVE_FRAMES = 10

# Initialize the frame counters for each action as well as
# booleans used to indicate if action is performed or not
MOUTH_COUNTER = 0
EYE_COUNTER = 0
WINK_COUNTER = 0
INPUT_MODE = False
EYE_CLICK = False
LEFT_WINK = False
RIGHT_WINK = False
SCROLL_MODE = False
```

```
ANCHOR_POINT = (0, 0)
WHITE_COLOR = (255, 255, 255)
YELLOW_COLOR = (0, 255, 255)
RED_COLOR = (0, 0, 255)
GREEN_COLOR = (0, 255, 0)
BLUE_COLOR = (255, 0, 0)
BLACK_COLOR = (0, 0, 0)

# Initialize Dlib's face detector (HOG-based) and then create
# the facial landmark predictor
shape_predictor = "model/shape_predictor_68_face_landmarks.dat"
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(shape_predictor)

# Grab the indexes of the facial landmarks for the left and
# right eye, nose and mouth respectively
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
(nStart, nEnd) = face_utils.FACIAL_LANDMARKS_IDXS["nose"]
(mStart, mEnd) = face_utils.FACIAL_LANDMARKS_IDXS["mouth"]

# Video capture
vid = cv2.VideoCapture(0)
resolution_w = 1366
resolution_h = 768
cam_w = 640
cam_h = 480
unit_w = resolution_w / cam_w
unit_h = resolution_h / cam_h

while True:
    # Grab the frame from the threaded video file stream, resize
    # it, and convert it to grayscale
    # channels)
    _, frame = vid.read()
    frame = cv2.flip(frame, 1)
    frame = imutils.resize(frame, width=cam_w, height=cam_h)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in the grayscale frame
```

```
rects = detector(gray, 0)

# Loop over the face detections
if len(rects) > 0:
    rect = rects[0]
else:
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
    continue

# Determine the facial landmarks for the face region, then
# convert the facial landmark (x, y)-coordinates to a NumPy
# array
shape = predictor(gray, rect)
shape = face_utils.shape_to_np(shape)

# Extract the left and right eye coordinates, then use the
# coordinates to compute the eye aspect ratio for both eyes
mouth = shape[mStart:mEnd]
leftEye = shape[lStart:lEnd]
rightEye = shape[rStart:rEnd]
nose = shape[nStart:nEnd]

# Because I flipped the frame, left is right, right is left.
temp = leftEye
leftEye = rightEye
rightEye = temp

# Average the mouth aspect ratio together for both eyes
mar = mouth_aspect_ratio(mouth)
leftEAR = eye_aspect_ratio(leftEye)
rightEAR = eye_aspect_ratio(rightEye)
ear = (leftEAR + rightEAR) / 2.0
diff_ear = np.abs(leftEAR - rightEAR)

nose_point = (nose[3, 0], nose[3, 1])

# Compute the convex hull for the left and right eye, then
# visualize each of the eyes
mouthHull = cv2.convexHull(mouth)
```

```
    leftEyeHull = cv2.convexHull(leftEye)
    rightEyeHull = cv2.convexHull(rightEye)
    cv2.drawContours(frame, [mouthHull], -1, YELLOW_COLOR, 1)
    cv2.drawContours(frame, [leftEyeHull], -1, YELLOW_COLOR, 1)
    cv2.drawContours(frame, [rightEyeHull], -1, YELLOW_COLOR, 1)

    for (x, y) in np.concatenate((mouth, leftEye, rightEye), axis=0):
        cv2.circle(frame, (x, y), 2, GREEN_COLOR, -1)

    # Check to see if the eye aspect ratio is below the blink
    # threshold, and if so, increment the blink frame counter
    if diff_ear > WINK_AR_DIFF_THRESH:

        if leftEAR < rightEAR:
            if leftEAR < EYE_AR_THRESH:
                WINK_COUNTER += 1
                cv2.putText(frame,    "Left    Click:"+str(WINK_COUNTER),    (250,    30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, GREEN_COLOR, 2)
                if WINK_COUNTER > WINK_CONSECUTIVE_FRAMES:
                    pag.click(button='left')


                    WINK_COUNTER = 0

        elif leftEAR > rightEAR:
            if rightEAR < EYE_AR_THRESH:
                WINK_COUNTER += 1
                cv2.putText(frame,    "Right    Click:"+str(WINK_COUNTER),    (250,    30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, GREEN_COLOR, 2)
                if WINK_COUNTER > WINK_CONSECUTIVE_FRAMES:
                    pag.click(button='right')

                    WINK_COUNTER = 0
        else:
            WINK_COUNTER = 0
    else:
        if ear <= EYE_AR_THRESH:
            EYE_COUNTER += 1
            cv2.putText(frame,    "Scroll    Mode:"+str(EYE_COUNTER),    (250,    30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, GREEN_COLOR, 2)  # Additional text
```

```
            if EYE_COUNTER > EYE_AR_CONSECUTIVE_FRAMES:
                SCROLL_MODE = not SCROLL_MODE
                # INPUT_MODE = not INPUT_MODE
                EYE_COUNTER = 0

                # nose point to draw a bounding box around it

        else:
            EYE_COUNTER = 0
            WINK_COUNTER = 0

    if mar > MOUTH_AR_THRESH:
        MOUTH_COUNTER += 1
        cv2.putText(frame,        "Mouth:"+str(MOUTH_COUNTER),       (250,        30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, GREEN_COLOR, 2)  # Additional text
        if MOUTH_COUNTER >= MOUTH_AR_CONSECUTIVE_FRAMES:
            # if the alarm is not on, turn it on
            INPUT_MODE = not INPUT_MODE
            # SCROLL_MODE = not SCROLL_MODE
            MOUTH_COUNTER = 0
            ANCHOR_POINT = nose_point

    else:
        MOUTH_COUNTER = 0

    if INPUT_MODE:
        cv2.putText(frame,         "READING         INPUT!",         (10,          30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, RED_COLOR, 2)

        x, y = ANCHOR_POINT
        nx, ny = nose_point
        w, h = 60, 35
        multiple = 1
        cv2.rectangle(frame, (x - w, y - h), (x + w, y + h), GREEN_COLOR, 2)
        cv2.line(frame, ANCHOR_POINT, nose_point, BLUE_COLOR, 2)

        dir = direction(nose_point, ANCHOR_POINT, w, h)
        cv2.putText(frame,  dir.upper(),  (10,  90),  cv2.FONT_HERSHEY_SIMPLEX,  0.7,
RED_COLOR, 2)
        drag = 18
        if dir == 'right':
            pag.moveRel(drag, 0)
        elif dir == 'left':
            pag.moveRel(-drag, 0)
        elif dir == 'up':
            if SCROLL_MODE:
                pag.scroll(40)
            else:
                pag.moveRel(0, -drag)
        elif dir == 'down':
            if SCROLL_MODE:
                pag.scroll(-40)
            else:
                pag.moveRel(0, drag)

    if SCROLL_MODE:
        cv2.putText(frame, 'SCROLL MODE IS ON!', (10, 60),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, RED_COLOR, 2)

    # cv2.putText(frame, "MAR: {:.2f}".format(mar), (500, 30),
    #           cv2.FONT_HERSHEY_SIMPLEX, 0.7, YELLOW_COLOR, 2)
    # cv2.putText(frame, "Right EAR: {:.2f}".format(rightEAR), (460, 80),
    #           cv2.FONT_HERSHEY_SIMPLEX, 0.7, YELLOW_COLOR, 2)
    # cv2.putText(frame, "Left EAR: {:.2f}".format(leftEAR), (460, 130),
    #           cv2.FONT_HERSHEY_SIMPLEX, 0.7, YELLOW_COLOR, 2)
    # cv2.putText(frame, "Diff EAR: {:.2f}".format(np.abs(leftEAR - rightEAR)), (460,
80),
    #           cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    # Show the frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # If the `Esc` key was pressed, break from the loop
    if key == 27:
        break

# Do a bit of cleanup
cv2.destroyAllWindows()
vid.release()
```

6.4 TEST CASES

Testing was carried out to evaluate the accuracy and stability of the Eye-Controlled Mouse using Facial Landmarks and Eye Gestures system. The goal of testing was to ensure that each gesture performed by the user is correctly recognized and that the corresponding mouse action is executed without delay. The system was tested using a standard 720p webcam in real-time under different lighting and environmental conditions.

Each test case was designed to verify a specific functionality such as facial landmark detection, eye gesture recognition, mouth-based input control, and nose-driven cursor movement. The results of these tests confirmed that the system performs accurately under normal conditions and provides reliable control of the mouse through facial gestures.

Table 6.1: Test Cases and Results

Description	Input / Action	Expected Output	Actual Output	Result
Face Detection	User sits in front of webcam	System detects face and landmarks	Face detected successfully	Pass
Mouth Open Detection	User opens mouth for 2–3 seconds	Input mode toggles (ON/OFF)	Mode switched correctly	Pass
Both Eyes Closed	User closes both eyes for 2 seconds	Scroll mode toggles (ON/OFF)	Scroll mode toggled	Pass
Left Eye Wink	User winks left eye	Left mouse click performed	Click executed correctly	Pass
Right Eye Wink	User winks right eye	Right mouse click performed	Click executed correctly	Pass
Nose Movement – Up	User moves head slightly upward	Cursor moves upward or scrolls up	Movement detected correctly	Pass
Nose Movement – Down	User moves head slightly downward	Cursor moves downward or scrolls down	Movement detected correctly	Pass
Poor Lighting Condition	Dim room light	Partial or no detection	Inconsistent detection	Fail / Partial
Multiple Faces	More than one person in frame	System detects a single active face	May detect incorrect user	Partial

Table 6.1: Test Cases for Eye Gesture Detection and Cursor Control

Chapter 7

Results and Discussion

The Eye-Controlled Mouse using Facial Landmarks and Eye Gestures system was implemented and tested to analyze its performance in real-time conditions. The primary goal of testing was to determine how effectively the system detects facial gestures and translates them into mouse actions such as cursor movement, clicking, and scrolling.

The system was evaluated under different lighting conditions, and multiple gestures were performed to test accuracy and responsiveness. The detection of facial landmarks using the Dlib predictor model was highly reliable, and the calculated eye and mouth aspect ratios (EAR and MAR) were effective in distinguishing between normal and active gestures.

The system performed well under standard lighting, achieving smooth and responsive control with minimal delay between gesture recognition and the corresponding mouse operation. However, detection accuracy decreased slightly in low-light environments or when multiple faces appeared in the frame. Overall, the results validate that the proposed system functions accurately, efficiently, and in real time, fulfilling its intended objectives.

Reading Input Mode:

In this stage, the system enters Input Mode when the user opens their mouth for a few seconds. Once activated, the cursor is controlled through the movement of the user's nose, which serves as a directional reference point. As the user moves their head, the cursor moves smoothly across the screen, allowing hands-free navigation.

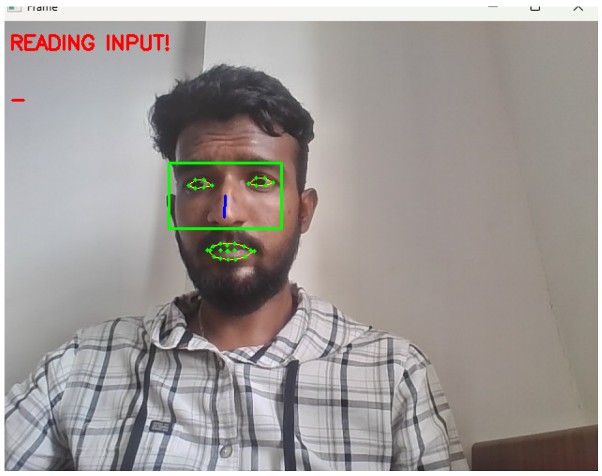

Figure 7.1: Reading Input Mode Activated

Left Click:

The left-eye wink gesture was tested to confirm the system's ability to detect partial eye closures. When the user winks with the left eye, the system interprets it as a left-click command and performs the action immediately using PyAutoGUI. The detection was consistent and precise across multiple trials.

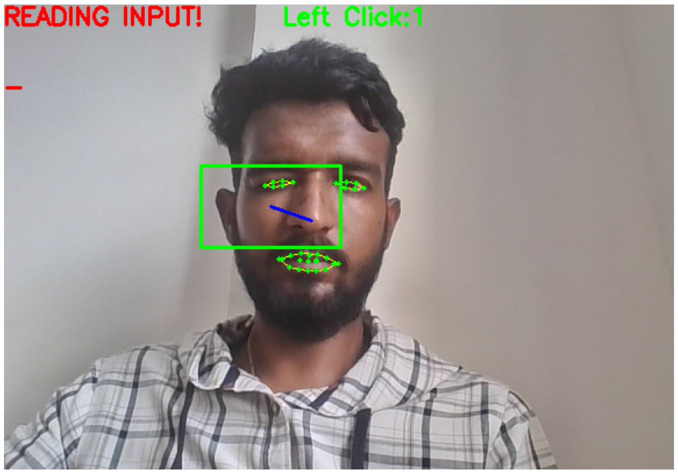

Figure 7.2: Left Eye Wink for Left Click

Right Eye Wink for Right Click

The right-eye wink was tested to verify differentiation between left and right gestures. When the right eye remains closed beyond the detection threshold, the system triggers a right-click operation. This feature enhances interactivity by allowing users to perform both primary and secondary click functions effortlessly through eye gestures.

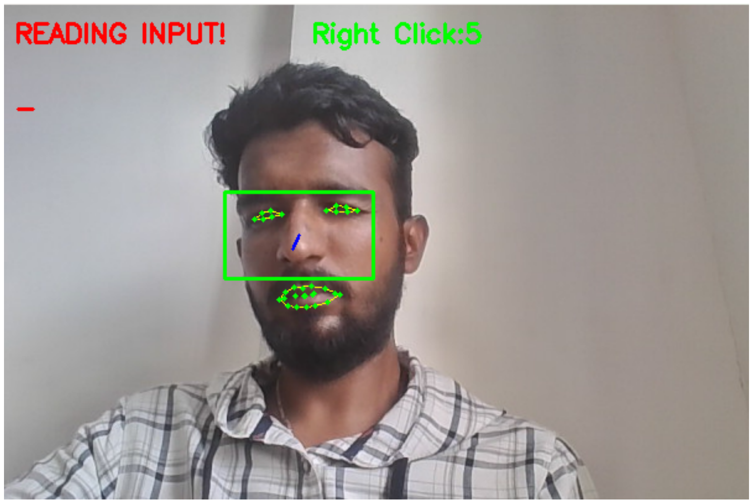

Figure 7.3: Right Eye Wink for Right Click

Scroll Mode Activated

This mode is enabled when the user closes both eyes for approximately two seconds. In Scroll Mode, vertical head movements control the scrolling action on the screen. Moving the head upward or downward scrolls the page accordingly. The system effectively detected this gesture and executed the scrolling action smoothly without lag.

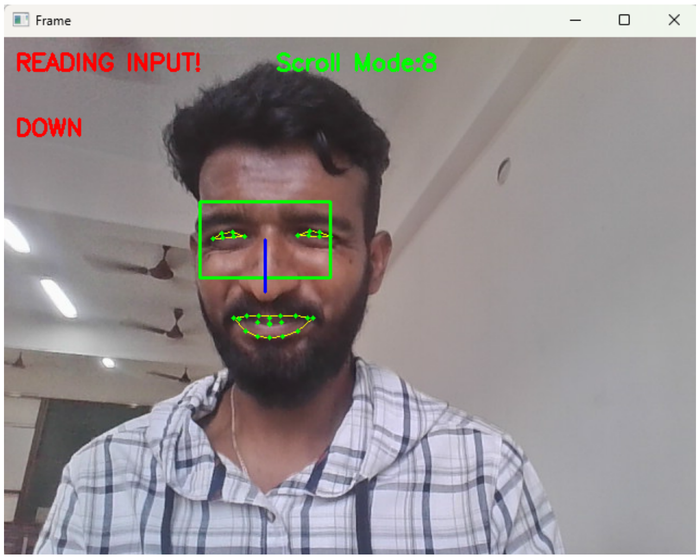

Figure 7.4: Scroll Mode Activated

The outputs confirm that the proposed system accurately identifies facial gestures and executes corresponding mouse actions in real time. The performance remains stable under proper lighting and single-user conditions, making it suitable for hands-free computer operation. Despite minor challenges in low-light scenarios, the system demonstrates a strong potential for practical use in assistive and human–computer interaction applications.

Chapter 8

Conclusion and Future Scope

The Eye-Controlled Mouse using Facial Landmarks and Eye Gestures system was successfully developed to provide a hands-free method of computer control using facial movements. The system effectively detects eye and mouth gestures through real-time image processing and translates them into mouse operations such as cursor movement, clicking, and scrolling.

Testing confirmed that the system performs with good accuracy and responsiveness under normal lighting conditions. The use of Python libraries like OpenCV, Dlib, and PyAutoGUI ensured efficient facial landmark detection and smooth interaction. The system demonstrated its potential as an assistive technology for individuals with limited hand mobility, offering an accessible alternative to traditional input devices.

Although the system performed well, it showed reduced accuracy in low-light environments and when multiple faces were visible. These limitations can be addressed in future versions by improving lighting adaptation, using high-resolution cameras, or integrating deep learning-based detection models.

Future enhancements may also include adding drag-and-drop functionality, gesture customization, and integration with voice commands to improve usability.

In conclusion, the project successfully achieved its objectives and proved that computer vision-based facial landmark tracking can be used effectively for real-time human–computer interaction and assistive applications.

Appendices

Appendix 1
SYSTEM SETUP AND CONFIGURATION

This appendix describes the hardware and software settings used during the implementation of the Eye-Controlled Mouse system.

Software Configuration:

Programming Language: Python 3.11

Libraries Used: OpenCV, Dlib, Imutils, NumPy, PyAutoGUI

Operating System: Windows 10

Model Used: shape_predictor_68_face_landmarks.dat (Dlib pre-trained facial landmark model)

Hardware Configuration:

Processor: Intel i5 / Ryzen 5 or higher

Memory: Minimum 4 GB RAM

Camera: Integrated 720p HD webcam

Display: 1080p monitor

Execution Steps:

1. Install all required Python libraries.
2. Place the Dlib landmark model in the project folder.
3. Run the script eye_controlled_mouse.py.
4. Allow webcam access and ensure sufficient lighting.
5. Perform eye gestures to control mouse operations.

Appendix 2
MATHEMATICAL FORMULATION

This appendix explains the formulas and parameters used to identify eye and mouth movements through facial landmarks.

1. Eye Aspect Ratio (EAR)

The Eye Aspect Ratio helps in determining whether the eye is open or closed based on vertical and horizontal distances between specific points surrounding the eye.

$$EAR = \frac{(\|p^2 - p^6\| + \|p^3 - p^5\|)}{(2 \times \|p^1 - p^4\|)}$$

Where:

- p1, p2, …, p6 represent the six coordinates located around the eye contour.
- || pi – pj|| denotes the Euclidean distance between the two points.
- A low EAR indicates a closed eye, whereas a high EAR indicates an open eye

2. Mouth Aspect Ratio (MAR)

The Mouth Aspect Ratio measures the vertical displacement of the lips relative to the mouth width, helping in detecting whether the mouth is open.

$$MAR = \frac{(\|p^{51} - p^{59}\| + \|p^{53} - p^{57}\| + \|p^{49} - p^{55}\|)}{3 \times \|p^{49} - p^{55}\|}$$

Where:

- p49, p51, p53, p55, p57, p59 denote the landmark coordinates surrounding the mouth region.

- A higher MAR value implies that the mouth is open.

Appendix 3
FUTURE ENHANCEMENT

- Possible improvements to enhance performance and adaptability of the system include:

- Introducing a drag-and-drop feature using extended eye closure.

- Implementing MediaPipe or deep learning frameworks for improved landmark accuracy.

- Integrating voice commands as a supplementary control option for accessibility.

References

[1] S. P. K. Mygapula, M. L. Saini, C. S. R. Dheeraj, S. Maji and D. Gupta, “Controlling Mouse Cursor through Eye Movement,” 2024 International Conference on Signal Processing and Advance Research in Computing (SPARC), Lucknow, India, 2024, pp. 1–5, doi: 10.1109/SPARC61891.2024.10828631.

[2] S. K. R., P. R., A. N. S. and M. Rathishree, “Cursor Control Based on Eyeball Movement Using Deep Learning,” 2023 Intelligent Computing and Control for Engineering and Business Systems (ICCEBS), Chennai, India, 2023, pp. 1–5, doi: 10.1109/ICCEBS58601.2023.10448773.

[3] R. H. Abiyev and M. Arslan, “Head Mouse Control System for People with Disabilities,” IEEE Access, 2019.

[4] V. S. Vasisht, S. Joshi, S. Shashidhar, S. Shreedhar and C. Gururaj, “Human Computer Interaction Based Eye Controlled Mouse,” Proceedings of the Third International Conference on Electronics Communication and Aerospace Technology (ICECA), 2019.

[5] M. Jayalakshmi, T. Pardha Saradhi, S. M. Rahil Azam, S. Durga Sai Sriram and S. Fazil, “Multi-model Human-Computer Interaction System with Hand Gesture and Eye Gesture Control,” 2024 5th International Conference on Innovative Trends in Information Technology (ICITIIT), 2024.

[6] S. Kanakaprabha, A. Prathibha, R. Priyanka and K. Vijay, “Mouse Cursor Controlled by Eye Movement for Individuals with Disabilities,” Proceedings of the 7th International Conference on Intelligent Computing and Control Systems (ICICCS), 2023.

[7] P. Miah, M. R. Gulshan and N. Jahan, “Mouse Cursor Movement and Control using Eye Gaze – A Human Computer Interaction,” 2022 International Conference on Artificial Intelligence of Things (ICAIoT), 2022.

[8] S. Sreeni, M. Sabeel, E. Sai Kumar, V. H. Vardhan and K. Chandrakala, “Mouse Cursor Control Using Facial Movements – An HCI Application,” International Journal of Techno-Engineering (IJTE), vol. XV, no. II, pp. 270–273, Apr. 2023.

[9] M. S. Murthy, D. Anitha, N. M. Chowdary and S. S. S. Kumar, “Mouse Cursor Control with Eye Gestures,” IEEE International Conference on Inventive Computation Technologies (ICICT), 2024.

[10] M. Dhanaraju, S. Mekala, A. H. V. Rao, C. P. Kumar and R. Lokesh, “Human-Eye Controlled Virtual Mouse,” International Journal for Research in Applied Science and Engineering Technology (IJRASET), vol. 10, no. VI, pp. 1100–1106, Jun. 2022.