A

PROJECT REPORT

ON

"GESTURE CONTROLLED VIRTUAL MOUSE"

Submitted by

1. Mr. Karmalkar Krushna Ganesh (2111100037)
2. Mr. Gaikwad Rutik Rajendra (2111100051)
3. Mr. Kulkarni Piyush Anand (2111100029)

UNDER THE GUIDANCE OF

Ms. Khatake S. R.

IN PARTIAL FULFILLMENT OF

DIPLOMA IN COMPUTER ENGINEERING

MAHARASTHRA STATE BOARD OF TECHNICAL EDUCATION

VIDYA PRATHISHTAN'S POLYTECHNIC COLLEGE,

INDAPUR-413106

VPP

DEPARTMENT OF COMPUTER ENGINEERING

Academic Year 2023-24

VPP

CERTIFICATE

This is to certify that the project entitled

"GESTURE CONTROLLED VIRTUAL MOUSE"

Submitted by

1. Mr. Karmalkar Krushna Ganesh (2111100037)
2. Mr. Gaikwad Rutik Rajendra (2111100051)
3. Mr. Kulkarni Piyush Anand (2111100029)

Has been successfully completed as per the requirements of the Maharashtra State Board of Technical Education, Mumbai in partial fulfilment of diploma in Computer Engineering for the academic year 2023-2024.

(Ms. Khatake S. R.)	**(Prof. Bhuse S. H.)**	**(Dr. Deshpande S. R.)**
PROJECT GUIDE	**HOD**	**PRINCIPAL**

VPP

CERTIFICATE

This is to certify that the project entitled

"GESTURE CONTROLLED VIRTUAL MOUSE"

Submitted by

1. Mr. Karmalkar Krushna Ganesh	(2111100037)
2. Mr. Gaikwad Rutik Rajendra	(2111100051)
3. Mr. Kulkarni Piyush Anand	(2111100029)

Has been successfully completed as per the requirements of the Maharashtra State Board of Technical Education, Mumbai in partial fulfilment of diploma in Computer Engineering for the academic year 2023-2024.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

After the successful implement of our project, we overcome with a sense of gratitude towards those people, without whose support, guidance and co-operation this would never have been possible.

First and foremost, we would like to thank our **H.O.D. Prof. Bhuse S. H.** and our **Project guide Ms. Khatake S. R.** for their valuable guidance which provided us with a perfect path on which we were able to successful implement our ideas.

We heartly like to thank our Principal **Dr. Deshpande S. R.** for their valuable support. Last but not least we would like to thank all our classmate & parents for their enthusiasm and great ideas.

Submitted by

1. Mr. Karmalkar Krushna Ganesh (2111100037)
2. Mr. Gaikwad Rutik Rajendra (2111100051)
3. Mr. Kulkarni Piyush Anand (2111100029)

Appendix-A

CERTIFICATE

This is to certify that,

Mr. Karmalkar Krushna Ganesh	(2111100037)
Mr. Gaikwad Rutik Rajendra	(2111100051)
Mr. Kulkarni Piyush Anand	(2111100029)

From Vidya Pratishthan's Polytechnic College, Indapur Institute have completed project of final year having title **Gesture Controlled Virtual Mouse** during the academic year 2023-2024. The project completed in a group consisting of 3 people under the guidance of the faculty guide.

Name and signature of guide: Ms. Khatake S. R.

Mobile number of Guide: +91 72189 65112

ABSTRACT

In this digital age, human-computer interaction has witnessed significant advancements, and one of the most exciting developments is the creation of a gesture-controlled virtual mouse system using Python. This project introduces a novel approach to navigate and interact with computers, offering a hands-free, intuitive, and futuristic interface for users. The system leverages computer vision and machine learning techniques, such as OpenCV and deep learning frameworks, to interpret hand gestures and convert them into mouse movements and actions. Through a webcam or depth-sensing camera, the system captures real-time hand movements and analyzes them to detect gestures and determine their spatial coordinates. The Python-based software translates these gestures into mouse events, including cursor movement, left and right-click actions, scrolling, and more. This opens up a new realm of possibilities for users, especially those with physical disabilities or in scenarios where traditional input devices may be impractical.

Content

1. Introduction

1.1 Introduction of Project:

In the rapidly evolving landscape of human-computer interaction, the integration of gesture control technology has emerged as a compelling avenue for enhancing user experiences. Traditional input devices such as keyboards and mice, while effective, often impose limitations on user mobility and interaction paradigms. Recognizing this, the project "Gesture Controlled Virtual Mouse" aims to revolutionize the way users interact with computers by harnessing the power of Python and a variety of libraries including TensorFlow, OpenCV, MediaPipe, and pyAutoGUI.

The project endeavors to create a virtual mouse system that allows users to control the cursor on their computer screen using hand gestures captured by a standard webcam. By leveraging the capabilities of machine learning, computer vision, and automation, this innovative solution seeks to provide a seamless and intuitive interface for navigating digital environments, interacting with applications, and manipulating on-screen elements.

At the heart of the project lies TensorFlow, a powerful open-source machine learning framework developed by Google. TensorFlow facilitates the training and deployment of deep learning models, making it an ideal choice for implementing gesture recognition algorithms. Through TensorFlow, the project aims to build a robust gesture recognition model capable of accurately interpreting hand movements captured by the webcam in real-time.

Complementing TensorFlow is OpenCV (Open Source Computer Vision Library), a popular open-source computer vision and image processing library. OpenCV provides a comprehensive set of tools and algorithms for tasks such as image manipulation, feature detection, and object tracking. In the context of the project, OpenCV plays a vital role in preprocessing the webcam feed, extracting relevant hand gestures, and performing real-time gesture recognition.

MediaPipe, another key component of the project, is a lightweight machine learning pipeline library developed by Google. MediaPipe offers a collection of pre-built models and tools for various multimedia processing tasks, including hand tracking and pose estimation. Leveraging MediaPipe's hand tracking capabilities, the project aims to precisely locate and track the user's hand movements within the webcam frame, facilitating accurate gesture recognition.

To bridge the gap between gesture recognition and system interaction, the project utilizes pyAutoGUI, a Python library for automating GUI interactions. pyAutoGUI enables programmatically controlling the mouse cursor, simulating keyboard input, and interacting with on-screen elements. By integrating pyAutoGUI with the gesture recognition pipeline, the project empowers users to navigate their computer interfaces effortlessly using hand gestures, effectively replacing the traditional mouse input.

The workflow of the Gesture Controlled Virtual Mouse project begins with capturing video input from the webcam using OpenCV. The webcam feed is then processed to detect and track the user's hand using MediaPipe's hand tracking model. Once the hand gestures are identified, TensorFlow comes into play, recognizing predefined gestures based on the trained machine learning model.

Upon recognizing a gesture, the corresponding action is triggered using pyAutoGUI, allowing the user to control the mouse cursor, perform clicks, scroll through content, and execute keyboard commands, all through intuitive hand movements. The system provides a seamless and intuitive user experience, enhancing accessibility and enabling users to interact with their computers in a more natural and efficient manner.

1.2 Problem Statement:

Developing a Gesture-Controlled Virtual Mouse using Python's TensorFlow, MediaPipe, OpenCV,PyAutoGUI, and related libraries aims to create an intuitive interface for computer interaction. The problem entails the need for a hands-free, intuitive, and efficient method to navigate computer interfaces, particularly beneficial for users with physical disabilities or situations where traditional input devices are impractical. The system aims to accurately track hand gestures in real-time using TensorFlow for deep learning-based hand pose estimation, MediaPipe for hand tracking, and OpenCV for computer vision tasks. By analyzing hand movements and gestures, the system must accurately interpret user intentions and translate them into corresponding mouse actions. Challenges include robustness against varying lighting conditions, occlusions, and diverse hand shapes and sizes. Additionally, the system should provide smooth and responsive interaction while minimizing latency. Integration with PyAutoGUI facilitates emulating mouse movements and clicks based on the detected gestures. Overall, this project addresses the need for a novel, hands-free input method through advanced machine learning and computer vision techniques, enhancing accessibility and user experience in human-computer interaction.

1.3 Scope of Project

The scope of the project "Gesture Controlled Virtual Mouse" aims to develop a robust and intuitive system utilizing Python's libraries such as MediaPipe, TensorFlow, PyAutoGUI, OpenCV, and others to create a virtual mouse interface controlled by hand gestures. The project will focus on enabling users to interact with their computers without physical contact with traditional input devices like a mouse or touchpad. The primary objective is to implement hand gesture recognition using MediaPipe and OpenCV to accurately detect and track hand movements in real-time video streams captured by a webcam. TensorFlow will be employed to train and deploy machine learning models for recognizing specific hand gestures, such as pointing, clicking, scrolling, and dragging. PyAutoGUI will be utilized to simulate mouse actions based on the recognized gestures, allowing users to control the cursor's movement, perform clicks, and execute other mouse operations. Additionally, the project will explore the integration of voice commands to complement gesture-based interactions, enhancing the system's accessibility and usability.

The project scope encompasses extensive testing and validation to ensure the accuracy, responsiveness, and reliability of the gesture recognition and virtual mouse functionalities across different lighting conditions, hand poses, and backgrounds. Performance optimization will also be a key focus to minimize latency and enhance the overall user experience. Furthermore, documentation will be provided to guide users on how to set up and use the gesture-controlled virtual mouse system effectively. The project will encourage community engagement and contributions by sharing the source code, tutorials, and resources to foster collaboration and further development in the field of human-computer interaction.

In conclusion, the project's scope is to develop an innovative and user-friendly solution for hands-free computer interaction through gesture-controlled virtual mouse technology, leveraging Python's libraries and machine learning techniques for seamless integration and functionality.

2. Literature Survey-Essential for finalizing problem Title.

Title: Virtual Mouse Using Hand Gesture

Author: E Sankar CHAVALI

Year: 2017

This project focuses on live video recording through a PC camera, recognizing hand gestures to trigger specific functionalities. The approach emphasizes real-time gesture recognition for practical applications

References:https://www.researchgate.net/publication/372165002_Virtual_Mouse_Using_Hand_Gesture

Title: Virtual Mouse using Hand Gestures

Author: Roshnee Matlani, Roshan Dadlani, Sharv Dumbre

Year: 2021

Matlani's study introduces a method for controlling the cursor's position without the need for electronic equipment. The system enables actions such as clicking and dragging through hand gestures, offering a hands-free alternative to traditional mouse interactions.

Reference: https://ieeexplore.ieee.org/document/9673251

Title: Virtual Mouse Using Hand Gesture

Author: Dubbaka Megha Sai Reddy, Srilekha Kukkamudi, Rishika Kunda

Year: 2011

Reddy's paper presents a camera vision-based cursor control framework capturing hand motions on a webcam. Python is utilized to implement the system, showcasing the versatility of programming languages in gesture-controlled interfaces.

Reference: https://ieeexplore.ieee.org/document/10060367

Title: Survey on Vision-based Hand Gesture Recognition

Author: Pranit Shah, Parul Universiy, Krishna Pandya, Harsh Shah, Jay Gandhi

Year: 2019

This survey paper demonstrates how computer vision techniques, utilizing the computer's window exploitation camera, can be employed to create a hand gesture-based virtual mouse, offering insights into the technology.

Reference:

https://www.researchgate.net/publication/335807462_Survey_on_Vision_based_Hand_Gesture_Recognition

Title: Gesture Controlled Virtual Mouse using AI

Author: Rekha B N, G Satish, Sampat Kundanagar, Vikyath Shetty

Year: 2013

This project offers a cursor control system with quick navigation of system controls using voice assistance and a camera for hand gesture recognition. It explores the integration of artificial intelligence in gesture-controlled virtual mouse systems.

Reference:https://www.ijraset.com/research-paper/gesture-controlled-virtual-mouse-using-ai

3. Scope of Project

3.1 Scope of Project:

The scope of the project "Gesture Controlled Virtual Mouse" aims to develop a robust and intuitive system utilizing Python's libraries such as MediaPipe, TensorFlow, PyAutoGUI, OpenCV, and others to create a virtual mouse interface controlled by hand gestures. The project will focus on enabling users to interact with their computers without physical contact with traditional input devices like a mouse or touchpad. The primary objective is to implement hand gesture recognition using MediaPipe and OpenCV to accurately detect and track hand movements in real-time video streams captured by a webcam. TensorFlow will be employed to train and deploy machine learning models for recognizing specific hand gestures, such as pointing, clicking, scrolling, and dragging. PyAutoGUI will be utilized to simulate mouse actions based on the recognized gestures, allowing users to control the cursor's movement, perform clicks, and execute other mouse operations. Additionally, the project will explore the integration of voice commands to complement gesture-based interactions, enhancing the system's accessibility and usability.

The project scope encompasses extensive testing and validation to ensure the accuracy, responsiveness, and reliability of the gesture recognition and virtual mouse functionalities across different lighting conditions, hand poses, and backgrounds. Performance optimization will also be a key focus to minimize latency and enhance the overall user experience. Furthermore, documentation will be provided to guide users on how to set up and use the gesture-controlled virtual mouse system effectively. The project will encourage community engagement and contributions by sharing the source code, tutorials, and resources to foster collaboration and further development in the field of human-computer interaction.

In conclusion, the project's scope is to develop an innovative and user-friendly solution for hands-free computer interaction through gesture-controlled virtual mouse technology, leveraging Python's libraries and machine learning techniques for seamless integration and functionality.

3.2 Prospects:

The future of gesture-controlled virtual mouse technology shows great promise, with various opportunities for advancement and application in diverse fields. Gesture-controlled interfaces offer a more intuitive and engaging interaction experience compared to traditional input methods. As technology evolves, improvements in gesture recognition algorithms and will further enhance the user experience, making interactions smoother and more immersive. Gesture-controlled virtual mouse systems can be enabling new possibilities for education, and productivity applications. This integration allows users to interact with digital content in three-dimensional space using intuitive hand gestures. Gesture-controlled interfaces can improve accessibility and inclusivity by providing alternative input methods for individuals with disabilities or mobility impairments.

As these technologies become more accessible and affordable, they can empower a wider range of users to access and interact with digital devices and applications. Furthermore, gesture-controlled virtual mouse applications can play a crucial role in enhancing productivity and efficiency in various industries and domains. By enabling hands-free interaction with digital tools and software applications, these applications can streamline workflow processes, improve collaboration, and increase overall productivity. For instance, in office environments, users can perform tasks such as navigating through documents, managing emails, or giving presentations without the need to physically touch a keyboard or mouse. Moreover, gesture-controlled virtual mouse applications have the potential to revolutionize accessibility for individuals with disabilities. By providing an alternative input method that does not rely on fine motor skills or physical dexterity, these applications can empower users with disabilities to navigate and interact with digital content more effectively. This inclusivity ensures that everyone, regardless of their abilities, can fully participate in the digital world.

3.3 Objective of project:

The objective of the project "Gesture Controlled Virtual Mouse" is to develop a novel and intuitive method for interacting with computers using hand gestures. Leveraging the capabilities of Python and various libraries such as TensorFlow, OpenCV, MediaPipe, and PyAutoGUI, this project aims to create a virtual mouse that can be controlled through hand movements captured by a webcam.

The traditional mouse and keyboard interface has long been the primary means of human-computer interaction, but it can be limiting in certain scenarios, such as when users have limited mobility or are unable to use conventional input devices. By introducing a gesture-controlled virtual mouse, this project seeks to provide an alternative interface that is more natural and accessible, allowing users to navigate and interact with applications using intuitive hand gestures.

The core technology behind the gesture-controlled virtual mouse relies on computer vision and machine learning techniques. OpenCV, a popular computer vision library, will be used to capture live video streams from a webcam and process them to detect and track the user's hand movements in real-time. MediaPipe, another powerful library, will facilitate hand landmark detection, enabling precise localization of key points on the hand, such as fingertips and palm.

TensorFlow, a leading machine learning framework, will play a crucial role in training and deploying a gesture recognition model. This model will be trained on a dataset of hand gesture samples, allowing it to accurately classify different gestures performed by the user. Through supervised learning techniques, the model will learn to associate specific hand gestures with corresponding mouse movements, such as cursor movement, clicking, scrolling, and dragging.

PyAutoGUI, a Python library for automating GUI interactions, will be utilized to simulate mouse actions based on the recognized gestures. By interfacing with the operating system's input controls, PyAutoGUI will enable the virtual mouse to translate detected hand gestures into corresponding mouse movements and clicks, effectively controlling the cursor and interacting with on-screen elements.

The overarching goal of the project is to create a robust and responsive gesture-controlled virtual mouse system that can seamlessly integrate with existing desktop applications and workflows. The system should be user-friendly and adaptable, allowing users to customize gestures, adjust sensitivity settings, and perform common tasks with ease. Additionally, the project aims to explore potential applications beyond traditional desktop computing, such as gaming, multimedia control, and accessibility solutions for users with disabilities.

By combining the capabilities of Python and various libraries such as TensorFlow, OpenCV, MediaPipe, and PyAutoGUI, this project seeks to push the boundaries of human-computer interaction, offering a more natural and intuitive way for users to interact with computers through hand gestures. Through experimentation, iteration, and user feedback, the project aims to refine and optimize the gesture-controlled virtual mouse system to deliver a seamless and immersive user experience.

4. Working and Specification of System, Software and Used Components

4.1 Software Requirement Specification:

- Python
- Gesture Recognition Libraries (OpenCV, MediaPipe, PyTorch or TensorFlow).
- GUI Library (Tkinter, PyQt or PyGTK)
- Mouse Emulation Libraries (PyAutoGUI, pynput)
- Operating System Compatibility
- IDE (Integrated Development Environment)- Visual Studio Code

4.2 Developers Requirement Hardware Requirements:

- Computer or Device
- Camera
- Input Device
- Power Supply
- Sufficient Lighting

4.3 Resources Required Application Requirements:

- Windows operating system
- Visual Studio IDE
- Python’s Library:
  - **i.** Tenserflow
  - **ii.** Mediapipe
  - **iii.** OpenCV
  - **iv.** PyAutoGUI
  - **v.** Math

4.4 About Tool Used:

The Gesture Controlled Virtual Mouse project is an innovative application that enables users to interact with their computers using hand gestures, effectively replacing traditional mouse input methods. Developed using Python and a variety of powerful libraries such as TensorFlow, OpenCV, MediaPipe, and PyAutoGUI, this project represents a cutting-edge fusion of computer vision, machine learning, and automation technologies.

At its core, the project utilizes the capabilities of TensorFlow, an open-source machine learning framework, to train a deep learning model for hand gesture recognition. By leveraging TensorFlow's flexibility and scalability, the model is trained to accurately identify and classify various hand gestures captured by the computer's camera.

OpenCV, a popular computer vision library, is instrumental in capturing real-time video streams from the computer's camera and performing image processing tasks such as hand detection and tracking. With its extensive collection of image processing algorithms and functions, OpenCV provides the necessary tools to preprocess video frames, extract relevant features, and detect the presence of hands in the camera feed.

MediaPipe, another powerful library, offers pre-built machine learning models and pipelines for various tasks, including hand pose estimation. By integrating MediaPipe's hand pose estimation model into the project, developers can accurately determine the precise positions and orientations of the user's hands in real-time, enabling robust gesture recognition and tracking.

PyAutoGUI, a cross-platform GUI automation library, serves as the bridge between gesture recognition and system interaction. Once a hand gesture is recognized and classified, PyAutoGUI translates the gesture into corresponding mouse movements and clicks, allowing users to control the computer's cursor and perform actions such as clicking, dragging, and scrolling without physical input devices.

The combination of these libraries enables the creation of a seamless and intuitive gesture-controlled interface for interacting with the computer. Users can perform a variety of gestures, such as pointing, swiping, and tapping, to navigate through applications, interact with user interfaces, and manipulate on-screen elements with ease.

One of the key advantages of this project is its versatility and adaptability to different environments and use cases. The modular architecture allows developers to customize and extend the functionality of the virtual mouse system according to specific requirements. Additionally, the use of Python and open-source libraries ensures that the project is accessible to a wide range of developers and researchers, fostering collaboration and innovation in the field of human-computer interaction.

In summary, the Gesture Controlled Virtual Mouse project represents a groundbreaking application of Python and various libraries such as TensorFlow, OpenCV, MediaPipe, and PyAutoGUI in the development of gesture-based interfaces. By harnessing the power of computer vision, machine learning, and automation, this project offers a novel and intuitive way for users to interact with their computers, paving the way for future advancements in human-computer interaction and user interface design.

4.5 Working of Components Required for project and its use:

1. **Camera:**
   - **Working:** A camera or sensor captures the user's hand gestures or movements.
   - **Use:** It translates physical gestures into digital signals that the computer can interpret for controlling the virtual mouse.

2. **Image Processing Library- OpenCV:**
   - **Working:** Processes the images or data captured by the camera/sensor.
   - **Use:** Detects and tracks the user's hand movements, recognizing specific gestures for controlling the virtual mouse.

3. **TensorFlow**:
   - **Working**: TensorFlow is used for training deep learning models to recognize hand gestures from input images or video frames.
   - **Use**: It enables the development of a model to classify different hand gestures, allowing the system to interpret user commands.

4. **PyAutoGUI**:
   - **Working**: PyAutoGUI automates tasks by controlling the mouse and keyboard, simulating mouse movements, clicks, and keystrokes.
   - **Use**: It moves the mouse cursor based on recognized hand gestures, creating a virtual mouse controlled by gestures.

5. **Mediapipe**:
   - **Working**: MediaPipe provides pre-built components for tasks like hand tracking, pose estimation, and object detection.
   - **Use**: It accurately tracks the user's hand position in real-time video streams, crucial for gesture recognition in the virtual mouse system.

6. **Machine Learning Models:**
   - **Working:** Trained models analyze the hand gesture data to recognize specific gestures.
   - **Use:** Enables the system to accurately interpret various hand gestures and translate them into corresponding mouse actions.

5. Methodology

The methodology for the project "Gesture Controlled Virtual Mouse" involves leveraging Python along with several key libraries such as TensorFlow, OpenCV, MediaPipe, and PyAutoGUI. Firstly, the project entails utilizing TensorFlow for training a machine learning model to recognize hand gestures accurately. OpenCV is then employed to capture real-time video input and perform hand detection and tracking. MediaPipe further enhances hand tracking accuracy and facilitates gesture recognition. Once gestures are detected, PyAutoGUI is utilized to translate these gestures into corresponding mouse movements and clicks. The methodology includes steps such as data collection, model training, real-time hand detection and tracking, gesture recognition, and virtual mouse control. By integrating these libraries effectively, the project aims to create an intuitive and responsive gesture-controlled virtual mouse system, providing users with an innovative and hands-free interaction experience.

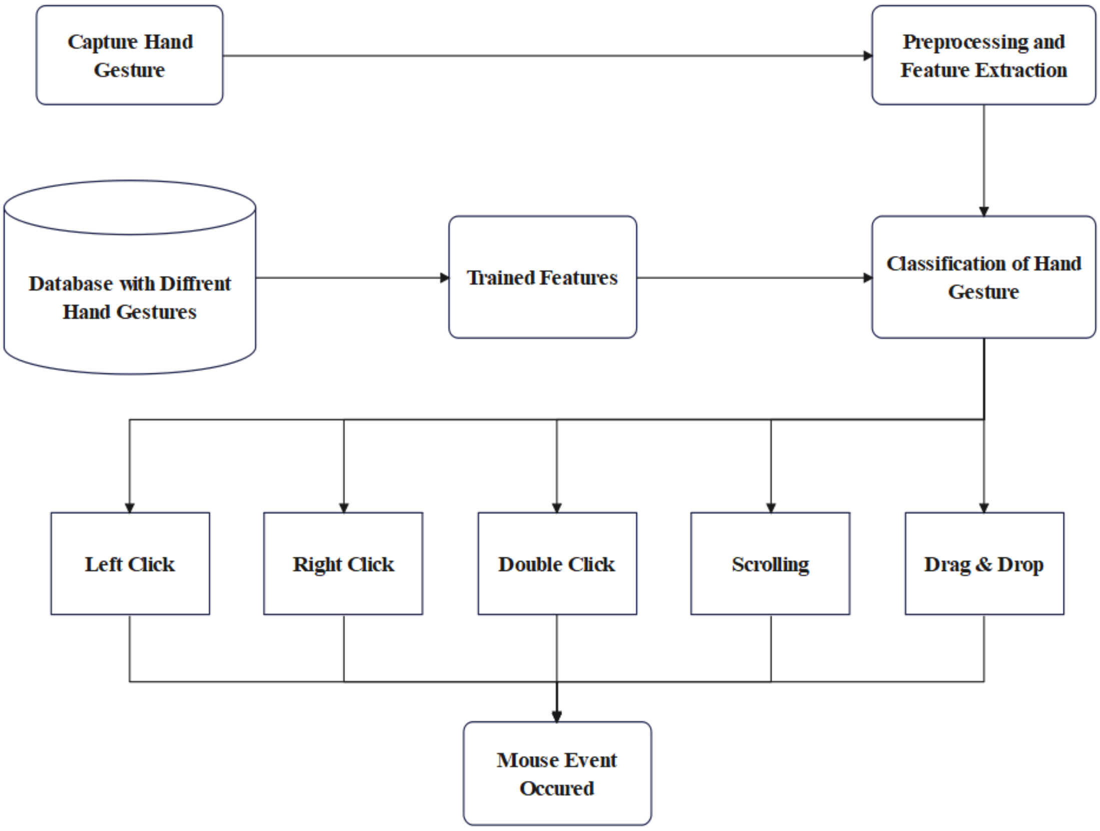
1. **Data Collection and Preparation:**
   a. **Data Sourcing:** Gather hand gesture datasets from open repositories like Kaggle or GitHub, or capture custom datasets using webcams and predefined gestures.

   b. **Data Diversity:** Ensure datasets encompass various hand shapes, sizes, skin tones, and lighting conditions to enhance model robustness and generalization.

2. **Model Development:**
   a. **Feature Extraction:** Utilizing computer vision algorithms, extract relevant features from input images, such as hand position, shape, and movement trajectories.

   b. **Model Training:** Train a machine learning model, possibly using deep learning techniques, on extracted features to accurately recognize and interpret hand gestures for controlling the virtual mouse.

- **Architecture:**



6. Project Life Cycle

6.1 Project Work Schedule (Timeline Chart):

Semester	Sem V													Sem VI											
Month	**Jul**	**Aug**			**Sep**			**Oct**			**Nov**			**Dec**			**Jan**			**Feb**			**Mar**		
Week	**4**	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
Information Gathering																									
Selection of Topic																									
Finalization of Topic																									
Synopsis																									
Requirement Analysis																									
Planning																									
Work Distribution																									
Modelling																									
Coding																									
Testing																									
Demonstration And Execution																									

Completed (Sem V)

Completed (Sem VI)

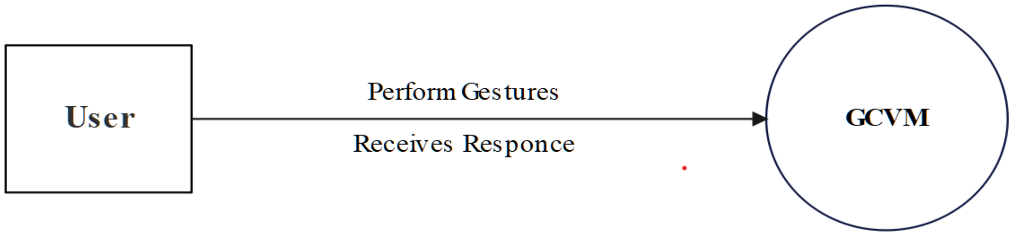
6.2 Time required for various stages for project:

Sr. No.	Work Planning	Working
4th week of July, 1st week of august	Requirement Gathering	In the 4th and 1st week of July & August, we started collecting information. This work is done by all the members together.
2nd, 3rd, week of august	Selection of Topic	Selection of Topic by different Domains takes place here.
1st,2nd week of September	Finalization of Topic.	Here we finalized our topic related to the respective domain
3rd week of Sep and 1st week of Oct	Synopsis	The whole documentation about the synopsis takes place here.
2nd & 3rd week of Oct	Requirement Analysis	Once the information is gathered, we had to do analysis of the requirements.
1st and 2nd week of Nov	Planning	Overall planning of project is done here.
3rd week of Nov	Work Distribution	The work is distributed to all members.
1st & 2nd week of Dec	Modelling	The project modelling and making the design got started.

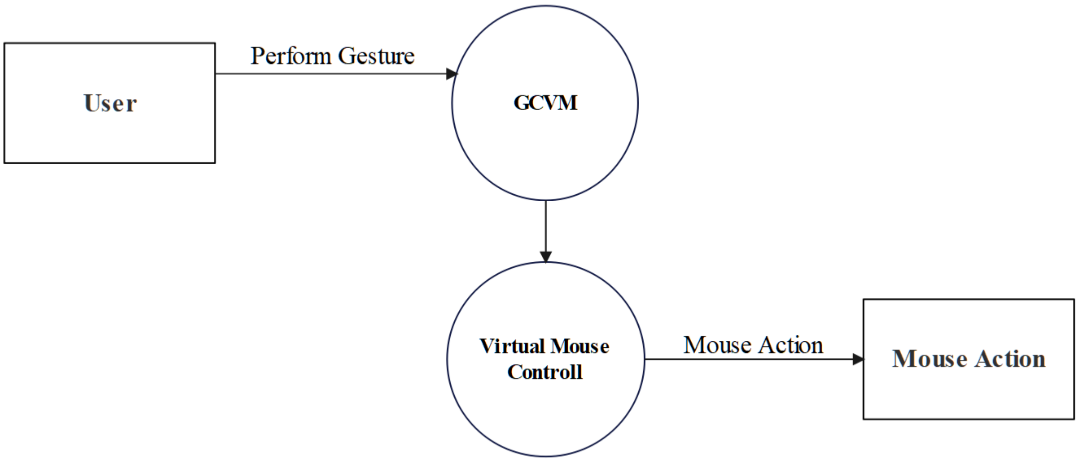
7. Design and working.

7.1 UML Diagrams:

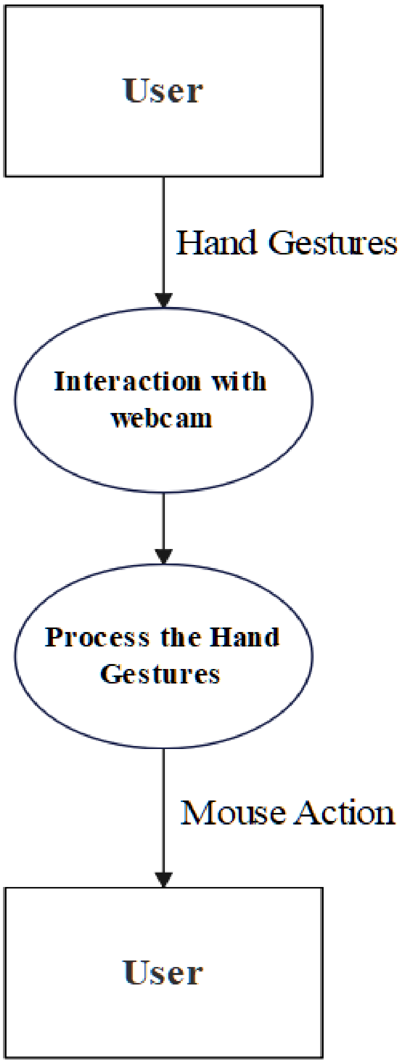
7.1.1 DFD Level 0:



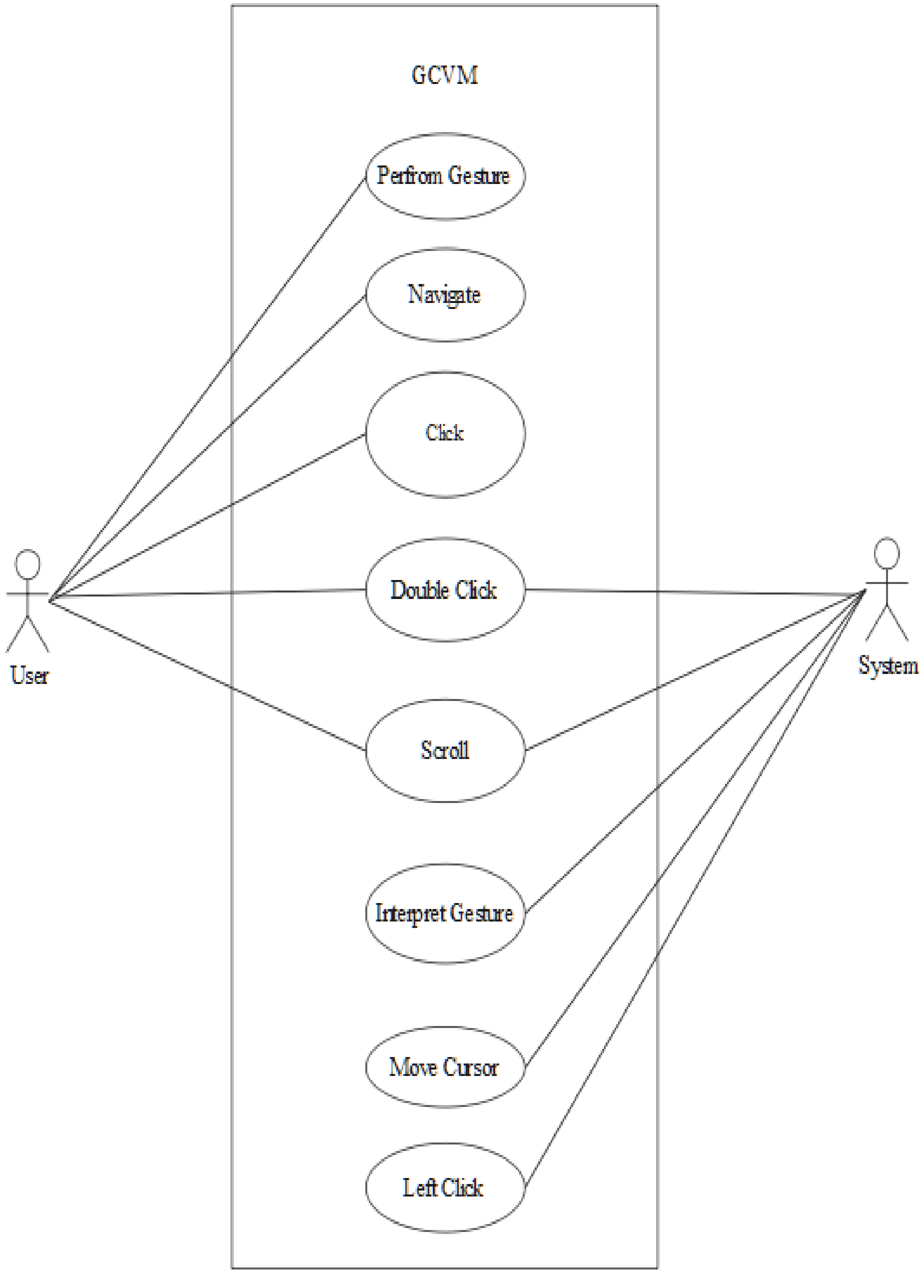
7.1.2 DFD Level 1:



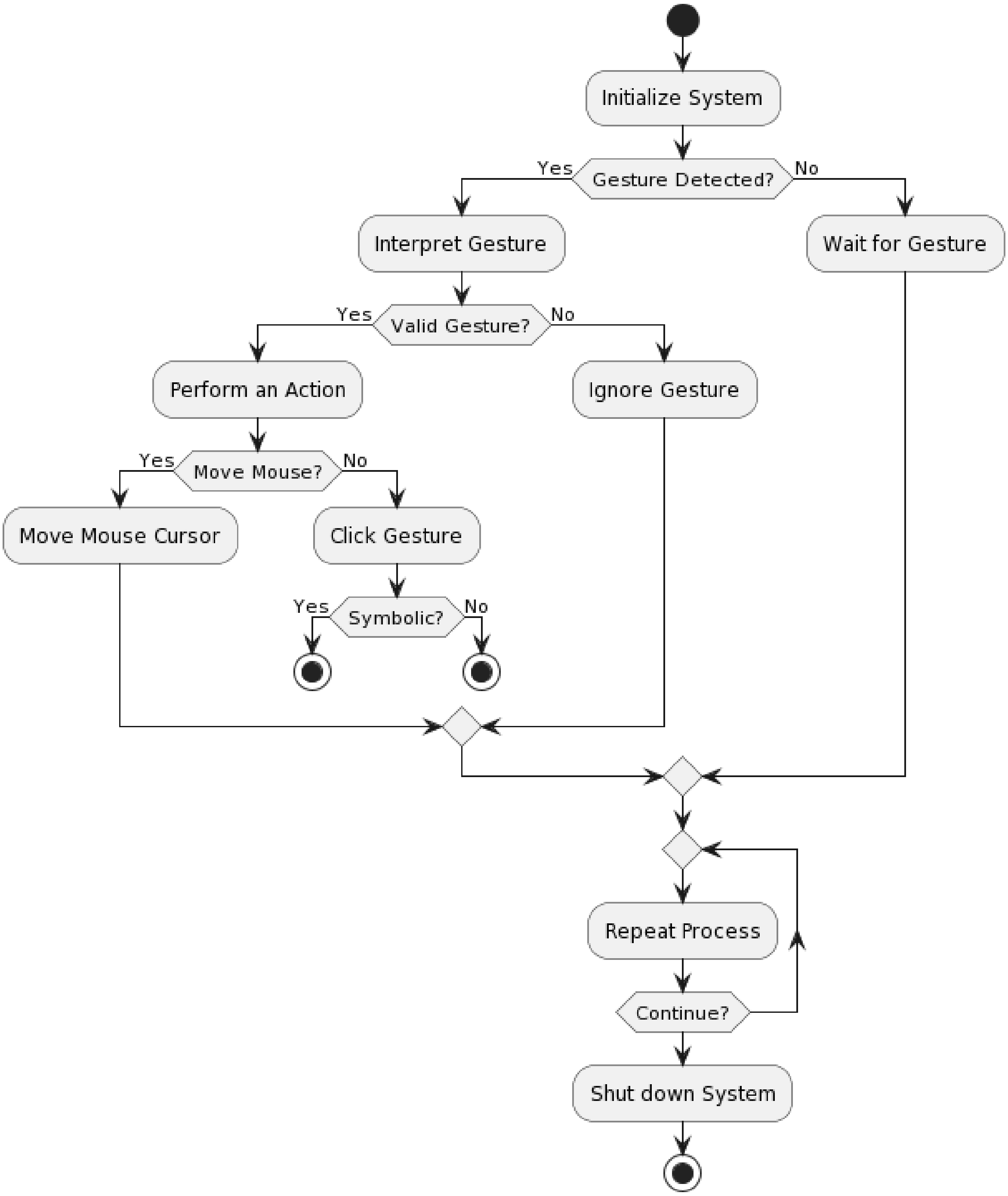
7.1.3 DFD Level 2:



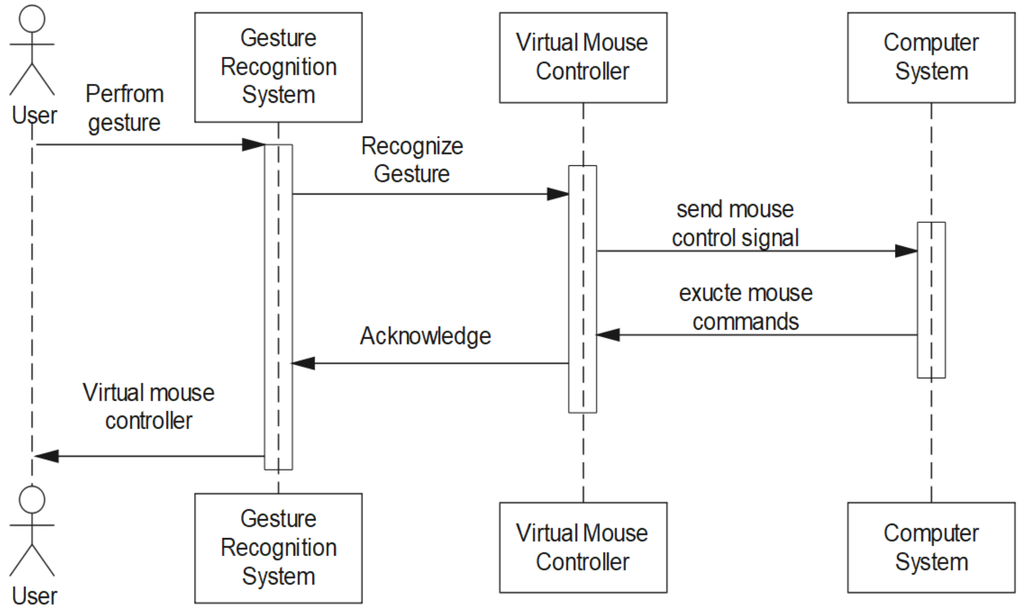
7.1.4 Use Case Diagram:



7.1.5 Activity Diagram:



7.1.6 Sequence Diagram:



8. Testing of Project

Test Cases:

Test Case ID	Test Case Name	Test Case Description	Input Data	Expected Output	Actual Output	Status
TC01	Detection of hand	To detect left or right hand	Hover the hand on web-cam	Hand is should be detected successfully	Hand is detected successfully	Pass
TC02	Multiple hand detection	To detect multiple hand (Right or Left)	Hover the hand on web-cam	Multiple hands should be detected	Multiple hands is detected	Pass
TC03	Multiple hand detection	To perform mouse event	Hover the hand on web-cam	Mouse event is occurred	Mouse is accessible For only one hand (Right)	Fail
TC04	Hand landmark	To detect hand with landmark	Hover the hand on web-cam	Hand is should be detected successfully With landmark	Hand is detected successfully With landmark	Pass
TC05	Right hand gesture	To perform mouse event using Right hand	Hover the hand on web-cam	Hand is should be detected successfully	Hand is detected successfully	Pass
TC06	Right hand gesture	To perform mouse event using Right hand	Hover the hand on web-cam	Hand is detected successfully	Error right hand is not detected	Fail
TC07	Left hand gesture	To perform mouse event using left hand	Hover the hand on web-cam	Hand should be detected & able to perform any mouse event	Hand is detected successfully but not perform any mouse event	Fail
TC08	Left hand gesture	To perform mouse event using	Hover the hand on web-cam	Hand is detected successfully	Error Left hand is not detected	Fail

		Left hand				
TC09	Left Click Gesture	To perform a left click gesture	Perform a left click gesture	Simulated left click event is triggered	Simulated left click event is triggered	Pass
TC10	Right Click Gesture	To perform a Right click Gesture	Perform a right click gesture	Simulated right click event is triggered	Simulated right click event is triggered	Pass
TC11	Move Mouse with V Gesture	To show hand with V gesture	Perform a V gesture to move the mouse cursor	Mouse cursor moves according to the gesture	Mouse cursor moves according to the gesture	Pass
TC12	Scroll with Pinch Gesture	To scroll with Pinch Gesture	Perform a pinch gesture for scrolling	Content on the screen scrolls up or down	Content on the screen scrolls up or down	Pass
TC13	Drag and Drop with Fist Gesture	To show hand for Drag an drop With fist Gesture	Perform a fist gesture for drag and drop action	Able to pick up and drop an object	Able to pick up and drop an object	Pass
TC14	No Gesture Registered	Gesture is not Registered	No gesture performed	No action is triggered	No action is triggered	Pass
TC15	Multiple Left Clicks	Try to left click For multiple times Using index finger	Rapidly perform multiple left click gestures	Each click is registered as individual clicks	Each click is registered as individual clicks	Pass
TC16	Long Press for Left Click	Long press for The left click Event	Long press with index finger for left click	Initiates a left click action after duration	Initiates a left click action after duration	Pass
TC17	Single Right Click	Try to single right click using Middle Finger	Perform a single right click gesture	Registers a single right click event	Registers a single right click event	Pass

TC18	Hold Right Click	Try to hold right click using Middle finger	Hold the middle finger gesture for an extended period	Continues to register right-click action	Continues to register right-click action	Pass
TC19	V Gesture Upwards	Try to v gesture upwards using Middle and index finger	Perform a V gesture upwards	Mouse cursor moves upward	Mouse cursor moves upward	Pass
TC20	V Gesture Downwards	Try to v gesture downwards using Middle and index finger	Perform a V gesture downwards	Mouse cursor moves downward	Mouse cursor moves downward	Pass
TC21	Pinch Gesture Upwards	Try to using pinch gesture for scroll up	Perform a pinch gesture upwards for scrolling	Content on the screen scrolls up	Content on the screen scrolls up	Pass
TC22	Pinch Gesture Downwards	Try to using pinch gesture for scrolls Down	Perform a pinch gesture downwards for scrolling	Content on the screen scrolls down	Content on the screen scrolls down	Pass
TC23	Drag Object with Fist Gesture	Try to fist gesture With drag and drop Event	Use a fist gesture to drag an object	Object follows the cursor movement	Object follows the cursor movement	Pass
TC24	Release Object with Fist Gesture	Try to release fist gesture For Dropped position	Release an object after dragging with a fist gesture	Object is dropped at the cursor position	Object is dropped at the cursor position	Pass
TC25	Invalid Gesture Combination	Any Invalid gesture Combination is Shown in camera	Perform an invalid combination of gestures	No action is triggered or error message displayed	No action is triggered or error message displayed	Pass

TC26	Simultaneous Gestures	Any simultaneous Gesture is shown in camera	Perform multiple gestures simultaneously	Each gesture is recognized independently	Each gesture is recognized independently	Pass
TC27	Swipe Gesture for Left Click	Try to swipe up gesture for Left click	Perform a swipe gesture for left-click action	Initiates a left click action in swipe direction	Initiates a left click action in swipe direction	Pass
TC28	Single swipe Down Gesture Recognition	Verify system Response to a Single swipe down Gesture	Gesture: Swipe down	Cursor moves Downwards On the screen.	Not moves a Cursor in downwards	Fail
TC29	Single swipe Up gesture Recognition	Verify system response to a Single swipe up Gesture	Gesture: Swipe up	Cursor Moves upwards On the screen	Not moves a cursor In upwards	Fail
TC30	Palm Gesture	Showing palm gesture to the camera	Show palm to the camera	Palm is detected	Palm is detected	Pass
TC31	Showing other Fingers	Showing any other Finger to the camera	Show any one finger of the camera	Finger is detected	Finger is detected	Pass
TC32	Left Click Success	Verify left-click event success	Perform left click	Simulated left-click event is triggered	Simulated left-click event	Pass
TC33	Left Click Fail	Verify left-click event failure	No input	No action is triggered	No action is triggered	Pass
TC34	Left Click with Index finger	Verify left-click with only Index finger	Perform gesture With only Index finger	Event trigged with Index finger	Event trigged	Pass

TC35	Left Click with multiple finger	Verify left-click with only Multiple finger	Perform gesture With Multiple finger	Event is not trigged	Event not trigged	Fail
TC36	Right Click Success	Verify right-click event success	Perform right click	Simulated right-click event is triggered	Simulated right-click event is triggered	Pass
TC37	Right Click Fail	Verify right-click event failure	No input	No action is triggered	No action is triggered	Pass
TC38	Right Click Middle Finger	Verify right-click with only Middle finger	Perform gesture with only Middle finger	Event triggered with Middle finger	Event triggered	Pass
TC39	Right Click Multiple Fingers	Verify right-click with multiple fingers	Perform gesture with multiple fingers	Event is not triggered	Event not triggered	Fail
TC40	Scroll Pinch Success	Verify scrolling with pinch gesture success	Perform pinch gesture	Content on the screen scrolls up or down	Content on the screen scrolls up or down	Pass

9. Cost Estimation

9.1 Bottom-Up approach cost estimation model:

We are completed this project in group of 3 members, this project contains 1 developer and 1 Tester & 1 Designer approximately we have completed the project in 9 months that means 210 days i.e., Total 434 hours. We have been working 14 hours a week on this project.

Role	Member Name	Monthly Salary	Daily Salary	Per hour
Developer	Karmalkar Krushna	5000	166	83
Tester	Gaikwad Rutik	3000	100	50
Designer	Kulkarni Piyush	3000	100	50
Total		11000	366	183

Total Monthly Salary for Developers, Testers and Designer:
Karmalkar K. G.= **5000 INR/month**
Gaikwad R. R.= **3000 INR/month,** Kulkarni P. A.= **3000 INR/month**
Total: 5000 + 3000 + 3000 = **11,000 INR/month**

Total Daily Salary for Developers, Testers and Designer:
Karmalkar K. G.: 5000 INR / 30 days =**166.67 INR/day**
Gaikwad R. R.: 3000 INR / 30 days = **100 INR/day,** Kulkarni P. A.= **100 INR/day**
Total: 166.67 + 100 + 100 = **366.67 INR/day**

Total Hourly Salary for Developers Testers and Designer:
Karmalkar K. G.: 166.67 INR / 2 hours = **83.33 INR/hour**
Gaikwad R. R.: 100 INR / 2 hours = **50 INR/hour**
Kulkarni P. A.= **50 INR/hour**
Total: 83.33 + 50 + 50 = **183.33 INR/hour**

Total Hours for the Project:
9 months * 30 days/month * 2 hours/day = **540 hours**

Total Cost for Developer, Tester and Designer (excluding profit and resources):
Hourly Salary * Total Hours = 183.33 INR/hour * 540 hours = **99,019.20 INR**

Adding 30% for Resources:
Total Cost * 30% = 99019.20 INR * 30% = **29,705.76 INR**
Adding 40% for Profit: - Total Cost + 40% = 99019.20 INR + 40% = **1,38,825.88 INR**

10. Coding

10.1 Coding for Gesture Controlled Virtual mouse:

main.py:

```
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
import tensorflow as tf
import cv2
import mediapipe as mp
import pyautogui
import math
from enum import IntEnum
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from google.protobuf.json_format import MessageToDict
import screen_brightness_control as sbcontrol

pyautogui.FAILSAFE = False
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands


class Gest(IntEnum):
    FIST = 0
    PINKY = 1
    RING = 2
    MID = 4
    LAST3 = 7
    INDEX = 8
    FIRST2 = 12
    LAST4 = 15
    THUMB = 16
    PALM = 31

    V_GEST = 33
    TWO_FINGER_CLOSED = 34
    PINCH_MAJOR = 35
    PINCH_MINOR = 36
```

```
class HLabel(IntEnum):
    MINOR = 0
    MAJOR = 1

class HandRecog:

    def __init__(self, hand_label):
        self.finger = 0
        self.ori_gesture = Gest.PALM
        self.prev_gesture = Gest.PALM
        self.frame_count = 0
        self.hand_result = None
        self.hand_label = hand_label

    def update_hand_result(self, hand_result):
        self.hand_result = hand_result

        sign = -1
        if self.hand_result.landmark[point[0]].y < self.hand_result.landmark[point[1]].y:
            sign = 1
        dist = (self.hand_result.landmark[point[0]].x -
self.hand_result.landmark[point[1]].x)**2
        dist += (self.hand_result.landmark[point[0]].y -
self.hand_result.landmark[point[1]].y)**2
        dist = math.sqrt(dist)
        return dist*sign

    def get_dist(self, point):

        dist = (self.hand_result.landmark[point[0]].x -
self.hand_result.landmark[point[1]].x)**2
        dist += (self.hand_result.landmark[point[0]].y -
self.hand_result.landmark[point[1]].y)**2
        dist = math.sqrt(dist)
        return dist

    def get_dz(self,point):

        return abs(self.hand_result.landmark[point[0]].z -
self.hand_result.landmark[point[1]].z)

    def set_finger_state(self):
```

```
        if self.hand_result == None:
            return

        points = [[8,5,0],[12,9,0],[16,13,0],[20,17,0]]
        self.finger = 0
        self.finger = self.finger | 0 #thumb
        for idx,point in enumerate(points):

            dist = self.get_signed_dist(point[:2])
            dist2 = self.get_signed_dist(point[1:])

            try:
                ratio = round(dist/dist2,1)
            except:
                ratio = round(dist/0.01,1)

            self.finger = self.finger << 1
            if ratio > 0.5 :
                self.finger = self.finger | 1

    def get_gesture(self):

        if self.hand_result == None:
            return Gest.PALM

        current_gesture = Gest.PALM
        if self.finger in [Gest.LAST3,Gest.LAST4] and self.get_dist([8,4]) < 0.05:
            if self.hand_label == HLabel.MAJOR :
                current_gesture = Gest.PINCH_MINOR
            else:
                current_gesture = Gest.PINCH_MAJOR

        elif Gest.FIRST2 == self.finger :
            point = [[8,12],[5,9]]
            dist1 = self.get_dist(point[0])
            dist2 = self.get_dist(point[1])
            ratio = dist1/dist2
            if ratio > 1.7:
                current_gesture = Gest.V_GEST
            else:
                if self.get_dz([8,12]) < 0.1:
                    current_gesture =  Gest.TWO_FINGER_CLOSED
                else:
```

```
                current_gesture =  Gest.MID

        else:
            current_gesture =  self.finger

        if current_gesture == self.prev_gesture:
            self.frame_count += 1
        else:
            self.frame_count = 0

        self.prev_gesture = current_gesture

        if self.frame_count > 4 :
            self.ori_gesture = current_gesture
        return self.ori_gesture

class Controller:
    tx_old = 0
    ty_old = 0
    trial = True
    flag = False
    grabflag = False
    pinchmajorflag = True
    pinchminorflag = False
    pinchstartxcoord = None
    pinchstartycoord = None
    pinchdirectionflag = None
    prevpinchlv = 0
    pinchlv = 0
    framecount = 0
    prev_hand = None
    pinch_threshold = 0.3

    def getpinchylv(hand_result):
        dist = round((Controller.pinchstartycoord - hand_result.landmark[8].y)*10,1)
        return dist

    def getpinchxlv(hand_result):
        dist = round((hand_result.landmark[8].x - Controller.pinchstartxcoord)*10,1)
        return dist

    def changesystembrightness():
        currentBrightnessLv = sbcontrol.get_brightness(display=0)/100.0
```

```
        currentBrightnessLv += Controller.pinchlv/50.0
        if currentBrightnessLv > 1.0:
            currentBrightnessLv = 1.0
        elif currentBrightnessLv < 0.0:
            currentBrightnessLv = 0.0
        sbcontrol.fade_brightness(int(100*currentBrightnessLv) , start =
sbcontrol.get_brightness(display=0))


    def changesystemvolume():

        devices = AudioUtilities.GetSpeakers()
        interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL,
None)
        volume = cast(interface, POINTER(IAudioEndpointVolume))
        currentVolumeLv = volume.GetMasterVolumeLevelScalar()
        currentVolumeLv += Controller.pinchlv/50.0
        if currentVolumeLv > 1.0:
            currentVolumeLv = 1.0
        elif currentVolumeLv < 0.0:
            currentVolumeLv = 0.0
        volume.SetMasterVolumeLevelScalar(currentVolumeLv, None)

    def scrollVertical():
        pyautogui.scroll(120 if Controller.pinchlv>0.0 else -120)


    def scrollHorizontal():

        pyautogui.keyDown('shift')
        pyautogui.keyDown('ctrl')
        pyautogui.scroll(-120 if Controller.pinchlv>0.0 else 120)
        pyautogui.keyUp('ctrl')
        pyautogui.keyUp('shift')


    def get_position(hand_result):

        point = 9
        position = [hand_result.landmark[point].x ,hand_result.landmark[point].y]
        sx,sy = pyautogui.size()
        x_old,y_old = pyautogui.position()
        x = int(position[0]*sx)
```

```
    y = int(position[1]*sy)
    if Controller.prev_hand is None:
        Controller.prev_hand = x,y
    delta_x = x - Controller.prev_hand[0]
    delta_y = y - Controller.prev_hand[1]

    distsq = delta_x**2 + delta_y**2
    ratio = 1
    Controller.prev_hand = [x,y]

    if distsq <= 25:
        ratio = 0
    elif distsq <= 900:
        ratio = 0.07 * (distsq ** (1/2))
    else:
        ratio = 2.1
    x , y = x_old + delta_x*ratio , y_old + delta_y*ratio
    return (x,y)

def pinch_control_init(hand_result):

    Controller.pinchstartxcoord = hand_result.landmark[8].x
    Controller.pinchstartycoord = hand_result.landmark[8].y
    Controller.pinchlv = 0
    Controller.prevpinchlv = 0
    Controller.framecount = 0

def pinch_control(hand_result, controlHorizontal, controlVertical):

    if Controller.framecount == 5:
        Controller.framecount = 0
        Controller.pinchlv = Controller.prevpinchlv

        if Controller.pinchdirectionflag == True:
            controlHorizontal() #x

        elif Controller.pinchdirectionflag == False:
            controlVertical() #y

    lvx =  Controller.getpinchxlv(hand_result)
    lvy =  Controller.getpinchylv(hand_result)

    if abs(lvy) > abs(lvx) and abs(lvy) > Controller.pinch_threshold:
```

```
            Controller.pinchdirectionflag = False
            if abs(Controller.prevpinchlv - lvy) < Controller.pinch_threshold:
                Controller.framecount += 1
            else:
                Controller.prevpinchlv = lvy
                Controller.framecount = 0

        elif abs(lvx) > Controller.pinch_threshold:
            Controller.pinchdirectionflag = True
            if abs(Controller.prevpinchlv - lvx) < Controller.pinch_threshold:
                Controller.framecount += 1
            else:
                Controller.prevpinchlv = lvx
                Controller.framecount = 0

    def handle_controls(gesture, hand_result):

        x,y = None,None
        if gesture != Gest.PALM :
            x,y = Controller.get_position(hand_result)

        if gesture != Gest.FIST and Controller.grabflag:
            Controller.grabflag = False
            pyautogui.mouseUp(button = "left")

        if gesture != Gest.PINCH_MAJOR and Controller.pinchmajorflag:
            Controller.pinchmajorflag = False

        if gesture != Gest.PINCH_MINOR and Controller.pinchminorflag:
            Controller.pinchminorflag = False

        if gesture == Gest.V_GEST:
            Controller.flag = True
            pyautogui.moveTo(x, y, duration = 0.1)

        elif gesture == Gest.FIST:
            if not Controller.grabflag :
                Controller.grabflag = True
                pyautogui.mouseDown(button = "left")
            pyautogui.moveTo(x, y, duration = 0.1)

        elif gesture == Gest.MID and Controller.flag:
            pyautogui.click()
```

```
            Controller.flag = False

        elif gesture == Gest.INDEX and Controller.flag:
            pyautogui.click(button='right')
            Controller.flag = False

        elif gesture == Gest.TWO_FINGER_CLOSED and Controller.flag:
            pyautogui.doubleClick()
            Controller.flag = False

        elif gesture == Gest.PINCH_MINOR:
            if Controller.pinchminorflag == False:
                Controller.pinch_control_init(hand_result)
                Controller.pinchminorflag = True
            Controller.pinch_control(hand_result,Controller.scrollHorizontal,
Controller.scrollVertical)

        elif gesture == Gest.PINCH_MAJOR:
            if Controller.pinchmajorflag == False:
                Controller.pinch_control_init(hand_result)
                Controller.pinchmajorflag = True
            Controller.pinch_control(hand_result,Controller.changesystembrightness,
Controller.changesystemvolume)

class GestureController:

    gc_mode = 0
    cap = None
    CAM_HEIGHT = None
    CAM_WIDTH = None
    hr_major = None
    hr_minor = None
    dom_hand = True

    def __init__(self):

        GestureController.gc_mode = 1
        GestureController.cap = cv2.VideoCapture(0)
        GestureController.CAM_HEIGHT =
GestureController.cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
        GestureController.CAM_WIDTH =
GestureController.cap.get(cv2.CAP_PROP_FRAME_WIDTH)
```

```
    def classify_hands(results):

        left , right = None,None
        try:
            handedness_dict = MessageToDict(results.multi_handedness[0])
            if handedness_dict['classification'][0]['label'] == 'Right':
                right = results.multi_hand_landmarks[0]
            else :
                left = results.multi_hand_landmarks[0]
        except:
            pass

        try:
            handedness_dict = MessageToDict(results.multi_handedness[1])
            if handedness_dict['classification'][0]['label'] == 'Right':
                right = results.multi_hand_landmarks[1]
            else :
                left = results.multi_hand_landmarks[1]
        except:
            pass

        if GestureController.dom_hand == True:
            GestureController.hr_major = right
            GestureController.hr_minor = left
        else :
            GestureController.hr_major = left
            GestureController.hr_minor = right

    def start(self):

        handmajor = HandRecog(HLabel.MAJOR)
        handminor = HandRecog(HLabel.MINOR)

        with mp_hands.Hands(max_num_hands = 2,min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
            while GestureController.cap.isOpened() and GestureController.gc_mode:
                success, image = GestureController.cap.read()

                if not success:
                    print("Ignoring empty camera frame.")
                    continue

                image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
```

```
            image.flags.writeable = False
            results = hands.process(image)

            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

            if results.multi_hand_landmarks:
                GestureController.classify_hands(results)
                handmajor.update_hand_result(GestureController.hr_major)
                handminor.update_hand_result(GestureController.hr_minor)

                handmajor.set_finger_state()
                handminor.set_finger_state()
                gest_name = handminor.get_gesture()

                if gest_name == Gest.PINCH_MINOR:
                    Controller.handle_controls(gest_name, handminor.hand_result)
                else:
                    gest_name = handmajor.get_gesture()
                    Controller.handle_controls(gest_name, handmajor.hand_result)

                for hand_landmarks in results.multi_hand_landmarks:
                    mp_drawing.draw_landmarks(image, hand_landmarks,
mp_hands.HAND_CONNECTIONS)
            else:
                Controller.prev_hand = None
            cv2.imshow('Gesture Controller', image)
            if cv2.waitKey(5) & 0xFF == 13:
                break
        GestureController.cap.release()
        cv2.destroyAllWindows()

gc1 = GestureController()
gc1.start()
```

GCVM.py:

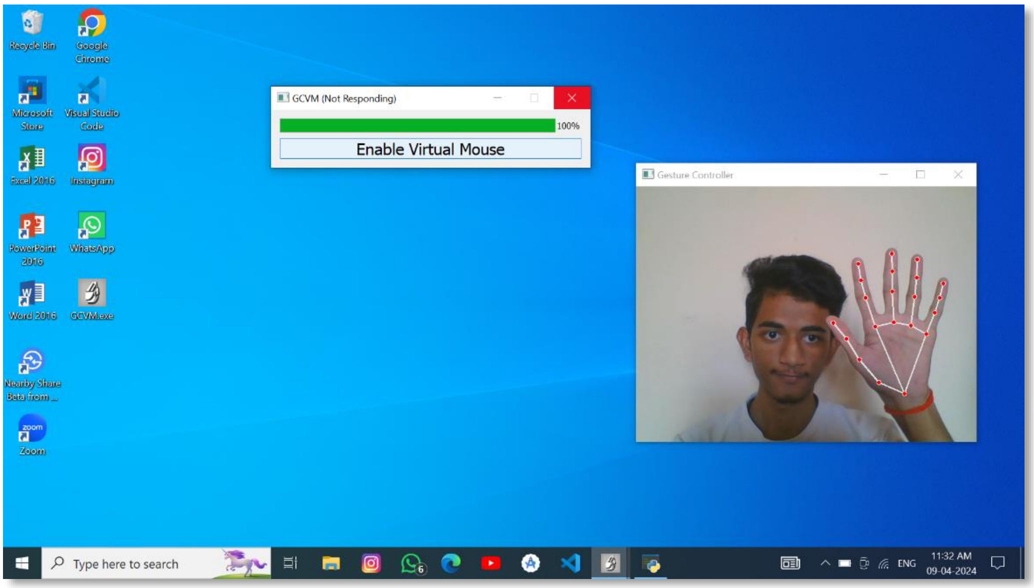
```
import sys
import time
import subprocess
from PyQt5.QtWidgets import QApplication, QWidget, QProgressBar, QPushButton,
QVBoxLayout
from PyQt5.QtCore import QThread, pyqtSignal, QTimer  # Import QTimer from
PyQt5.QtCore


class WorkerThread(QThread):
    progress_signal = pyqtSignal(int)

    def __init__(self, script_path, n):
        super().__init__()
        self.script_path = script_path
        self.n = n

    def run(self):
        for i in range(self.n):
            time.sleep(0.01)
            self.progress_signal.emit(i + 1)

        # Execute the Python script using subprocess
        try:
            subprocess.run(['python', self.script_path])
        except subprocess.CalledProcessError as e:
            print(f"Error executing script: {e}")

        # You can add additional cleanup or handling after the subprocess is complete


class AppDemo(QWidget):
    def __init__(self):
        super().__init__()
        self.setMinimumWidth(600)
        self.setWindowTitle('GCVM')  # Set the window title to 'GCVM'

        layout = QVBoxLayout()

        self.progressBar = QProgressBar()
```

```
        self.progressBar.setMinimum(0)
        layout.addWidget(self.progressBar)

        self.button = QPushButton('Enable Virtual Mouse')
        self.button.setStyleSheet('font-size: 30px; height: 30px;')
        self.button.clicked.connect(self.start_subprocess)

        layout.addWidget(self.button)
        self.setLayout(layout)

        self.worker_thread = None
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.update_gui)

    def start_subprocess(self):
        script_path = 'C:\\GCVM\\main.py'
        n = 500
        self.progressBar.setValue(0)

        # Start the subprocess in a separate thread
        self.worker_thread = WorkerThread(script_path, n)
        self.worker_thread.progress_signal.connect(self.update_progress)
        self.worker_thread.start()

        # Start the timer to periodically check for events
        self.timer.start(10)

    def update_progress(self, value):
        self.progressBar.setValue(value)

        # If subprocess is complete, stop the thread and timer
        if value == self.progressBar.maximum():
            self.worker_thread.quit()
            self.worker_thread.wait()
            self.timer.stop()

    def update_gui(self):
        # Periodically check for events and update the GUI
        QApplication.processEvents()
```

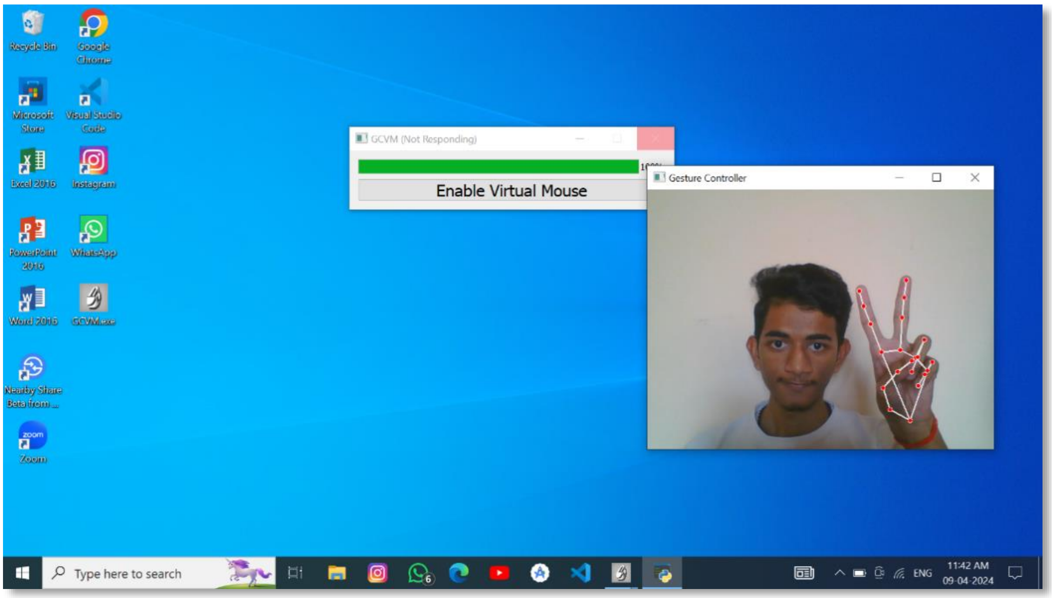
```
if __name__ == '__main__':
    app = QApplication(sys.argv)

    demo = AppDemo()
    demo.show()

    try:
        sys.exit(app.exec_())
    except SystemExit:
        print('Closing Window...')
```

11. Result and Applications

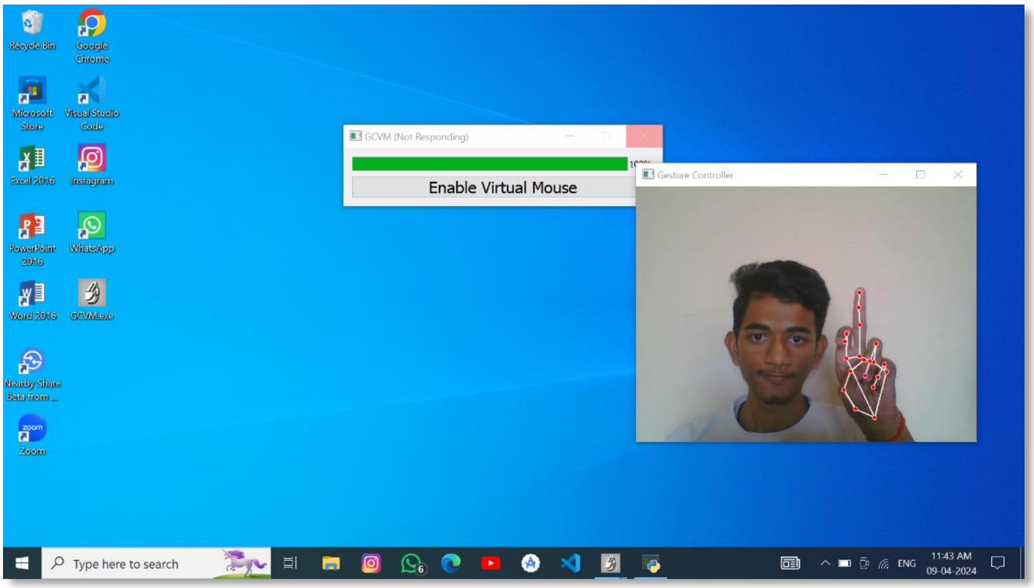
11.1 Neutral Gesture:



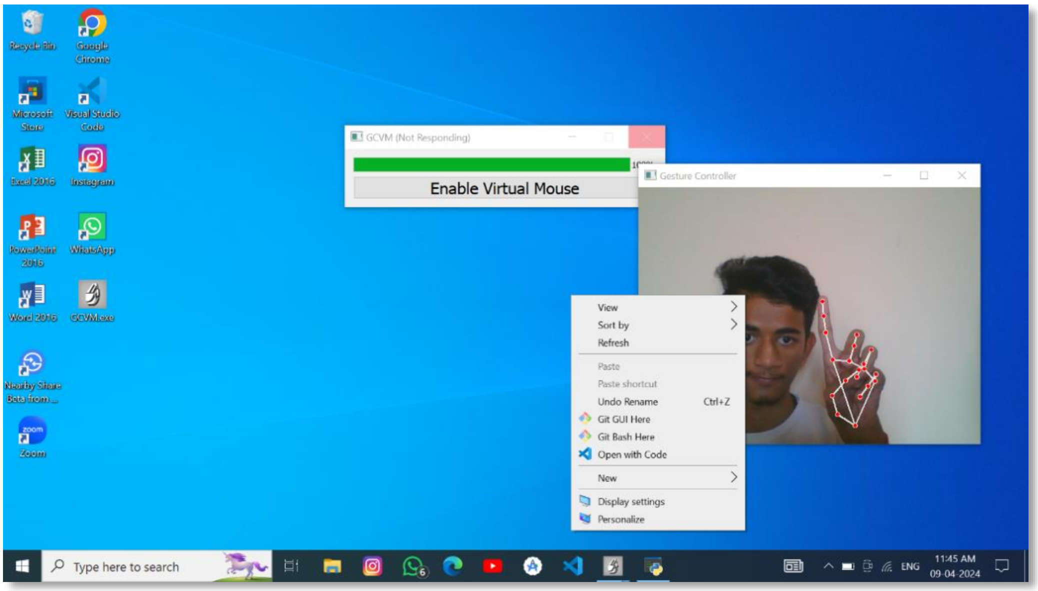
11.2 Cursor Movement Gesture:



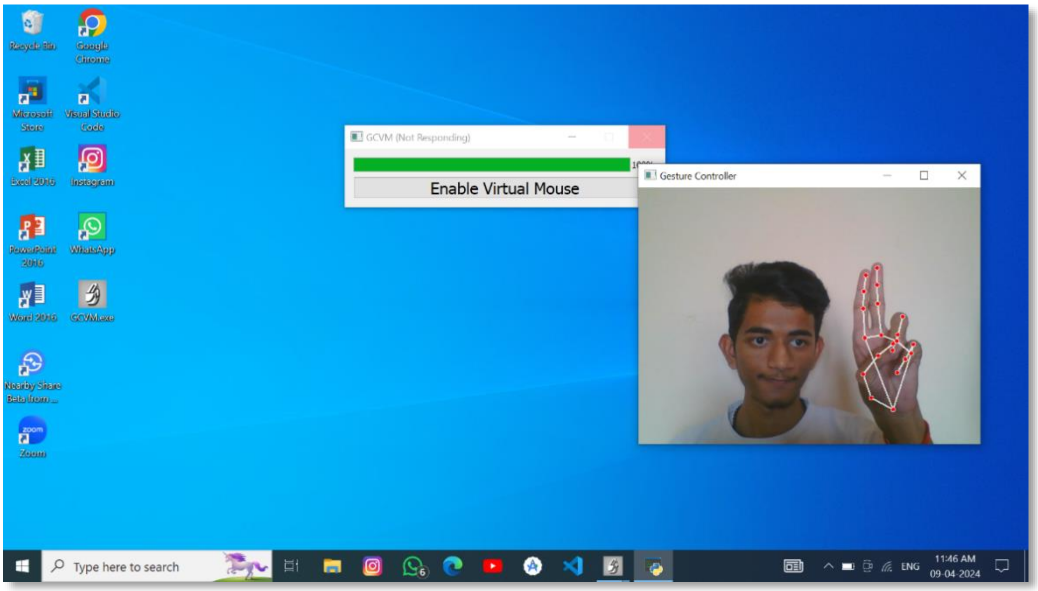
11.3 Left Click Gesture:



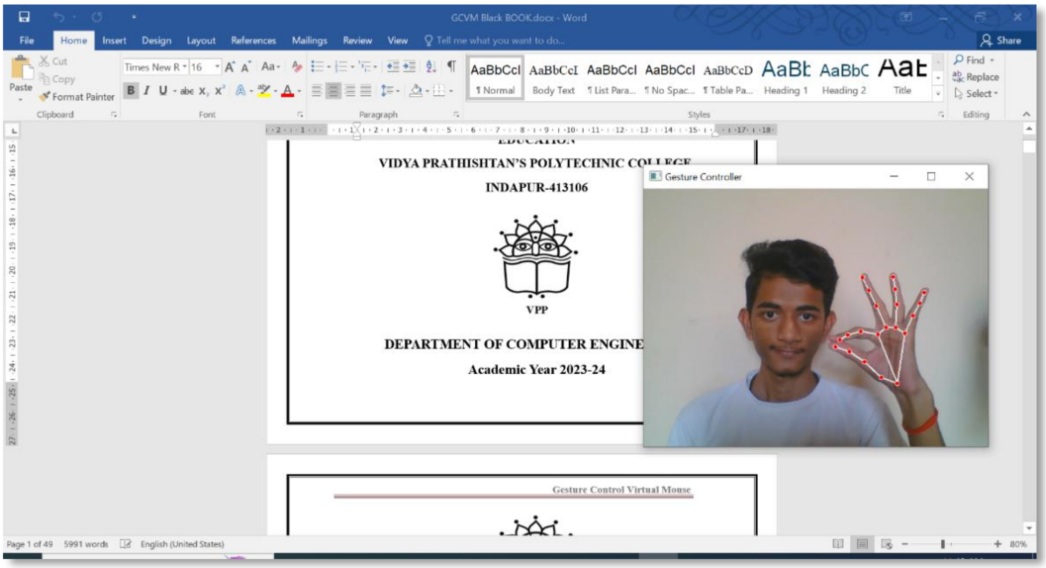
11.4 Right Click Gesture:



11.5 Double Click Gesture:



11.6 Scrolling Gesture:



11.7 Applications:

1. **Presentation Control:** Navigate slideshows with hand gestures, providing seamless interaction during presentations without needing to rely on traditional input devices.

2. **Media Player Control:** Manage playback of multimedia content (e.g., videos, music) using intuitive hand gestures, enhancing user experience and accessibility.

3. **Gaming Interaction:** Integrate gesture-based controls into games for immersive experiences, offering a novel way to interact with virtual environments.

4. **Computer Accessibility:** Enable individuals with mobility impairments to navigate and control computers using hand gestures, fostering inclusivity in computing.

5. **Virtual Reality Navigation:** Enhance VR experiences by allowing users to navigate menus, interact with objects, and control movement through gestures, eliminating the need for physical controllers.

6. **Educational Tools:** Develop interactive educational applications where students can manipulate virtual objects and interfaces using hand gestures, promoting engagement and learning retention.

12. Conclusion (with respect to POs)

- We successfully demonstrated the feasibility of using computer vision techniques to translate hand gestures into mouse commands.

- Through the utilization of Python and its libraries, we have developed a functional system that allows users to interact with digital interfaces intuitively.

- This project not only showcases technical proficiency but also underscores the importance of interdisciplinary collaboration and effective communication.

- As we continue to refine and optimize our solution, we remain committed to exploring innovative ways to enhance human-computer interaction using Python and its diverse ecosystem of tools and libraries.

13. Evolution of Project

13.1 Memory Analysis:

Memory analysis is a pivotal aspect of software development and optimization, aimed at maximizing the efficient utilization of memory resources within a computer system. This process entails a thorough examination of memory allocation, usage, and deallocation throughout program execution to pinpoint potential issues such as memory leaks, excessive consumption, and suboptimal management practices. By engaging in memory analysis, developers can glean valuable insights into the memory behavior of their software applications, enabling them to address memory-related issues that might otherwise compromise performance, system stability, or security.

Memory analysis encompasses a range of tools and techniques that empower developers to monitor memory usage in real-time, scrutinize allocation patterns, and detect memory leaks by identifying unreferenced memory blocks that linger without proper deallocation. These tools provide developers with a comprehensive view of memory utilization, aiding in the identification of inefficiencies and areas for improvement. Ultimately, memory analysis is instrumental in ensuring the reliability, efficiency, and stability of software applications, particularly in environments characterized by intense memory demands such as embedded systems, mobile devices, and high-performance computing platforms.

The primary objective of memory analysis is to optimize memory usage, enhance application performance, and elevate the overall user experience. By identifying and rectifying memory-related issues early in the development cycle, developers can preemptively mitigate potential performance bottlenecks, system crashes, or security vulnerabilities. Additionally, efficient memory management translates to optimized resource utilization, enabling applications to run more smoothly and reliably across diverse computing environments.

One of the key challenges in memory analysis lies in effectively identifying and addressing memory leaks, which occur when allocated memory is no longer accessible but remains allocated, thereby progressively consuming system resources. Memory leaks can lead to degraded performance over time and may eventually result in system instability or crashes. Memory analysis tools equipped with advanced algorithms and heuristics play a crucial role in detecting these leaks, allowing developers to pinpoint their origins and implement appropriate fixes.

Furthermore, memory analysis facilitates the identification of memory fragmentation, a phenomenon where memory becomes divided into smaller, unusable chunks over time, diminishing the overall efficiency of memory allocation. By analyzing memory allocation patterns and fragmentation levels, developers can implement strategies to mitigate fragmentation and optimize memory utilization.

In conclusion, memory analysis serves as a cornerstone of software development and optimization, enabling developers to proactively identify and address memory-related issues that could impair application performance, stability, or security. By leveraging memory analysis tools and techniques, developers can optimize memory usage, enhance application performance, and deliver a superior user experience across diverse computing environments. Ultimately, investing in robust memory analysis practices is essential for ensuring the reliability, efficiency, and scalability of software applications in today's increasingly memory-intensive computing landscape.

13.2 Platform:

The platform serves as the foundational infrastructure for software applications, encompassing both hardware and software components essential for their operation. It comprises the operating system, hardware architecture, programming languages, libraries, and frameworks upon which applications are built and executed. A thorough understanding of the platform is paramount for developers to ensure compatibility, performance optimization, and portability across various devices and environments. When developing software, developers need to account for the specific characteristics and constraints of the platform they are targeting.

Different platforms may have unique requirements, capabilities, and limitations that influence the design and implementation of software solutions. For instance, considerations such as memory management, processing power, and input/output mechanisms may vary significantly between platforms. By comprehending these platform-specific nuances, developers can tailor their applications to leverage platform features effectively while mitigating potential compatibility issues or performance bottlenecks.

Moreover, platform selection plays a pivotal role in shaping deployment strategies and user experience. Developers must choose the most suitable platform based on factors such as target audience, device compatibility, market trends, and development resources. For instance, mobile applications may need to be developed for specific operating systems like iOS or Android, each with its own set of development tools and design guidelines.

By aligning with the chosen platform's conventions and best practices, developers can create applications that seamlessly integrate with the user's device environment, thereby enhancing user satisfaction and engagement. Furthermore, understanding the platform enables developers to optimize software performance and resource utilization. By leveraging platform-specific features, APIs, and optimization techniques, developers can enhance application efficiency and responsiveness. For example, applications designed for cloud platforms may utilize scalable infrastructure services to dynamically adjust resource allocation based on demand, optimizing cost-efficiency and scalability.

Additionally, platform awareness extends beyond the development phase and influences the entire software development lifecycle. Maintenance, updates, and support activities are inherently tied to the underlying platform, necessitating ongoing consideration of platform compatibility and evolving requirements. As platforms evolve over time, developers must adapt their software accordingly to maintain compatibility and leverage new features or enhancements introduced by the platform provider.

In conclusion, understanding the platform is fundamental for developing robust, efficient, and compatible software applications. Developers must navigate platform-specific requirements, constraints, and opportunities to ensure seamless operation, optimal performance, and a positive user experience across diverse devices and environments. By embracing platform awareness throughout the software development lifecycle, developers can create software solutions that effectively meet user needs while staying adaptable to evolving platform landscapes.

13.3 Performance:

The project "Gesture Controlled Virtual Mouse" utilizes Python along with several libraries such as TensorFlow, OpenCV, MediaPipe, and PyAutoGUI to create an innovative solution that allows users to control their computer cursor through hand gestures. This project leverages machine learning and computer vision techniques to interpret hand movements and translate them into mouse actions, offering a novel and intuitive way to interact with computers.

At its core, TensorFlow is utilized for training and deploying machine learning models that can recognize and classify hand gestures from video input. TensorFlow's powerful neural network framework enables the creation of custom models tailored to the specific gestures required for controlling the virtual mouse. Through training on labeled gesture data, the model learns to accurately identify different hand movements, forming the basis for gesture recognition in real-time.

OpenCV (Open Source Computer Vision Library) plays a pivotal role in processing video streams from the computer's webcam or external camera. It provides a suite of functions for image manipulation, feature detection, and object tracking, making it an ideal choice for capturing and analyzing live video input. OpenCV's capabilities are instrumental in detecting and tracking the user's hand movements within the video feed, facilitating the extraction of relevant features for gesture recognition.

MediaPipe, a machine learning framework developed by Google, complements OpenCV by offering pre-trained models and pipelines for hand detection and pose estimation. By leveraging MediaPipe's hand tracking models, the project can accurately locate and extract key points corresponding to the user's hand joints and fingertips. This data is then utilized to infer hand gestures and translate them into corresponding mouse actions.

PyAutoGUI serves as the interface between the gesture recognition system and the computer's operating system, enabling the emulation of mouse movements, clicks, and other interactions. With PyAutoGUI, the project can simulate mouse movements based on the recognized gestures, allowing users to navigate and interact with applications using intuitive hand gestures. PyAutoGUI's cross-platform support ensures compatibility with various operating systems, making the virtual mouse accessible to a wide range of users.

Overall, the "Gesture Controlled Virtual Mouse" project showcases the potential of Python and its libraries for creating innovative human-computer interaction solutions. By combining machine learning, computer vision, and automation techniques, the project offers a seamless and intuitive alternative to traditional mouse input methods. With its versatility and adaptability, this project has the potential to revolutionize the way users interact with computers, opening up new possibilities for accessibility, productivity, and user experience enhancement.

13.4 Advantages:

a. **Intuitive Interaction:** Users can navigate and interact with digital interfaces using natural hand gestures, providing a more intuitive and immersive user experience.

b. **Touchless Operation**: Eliminates the need for physical contact with input devices, promoting hygiene and reducing the risk of germ transmission, especially in shared or public environments.

c. **Accessibility:** Offers an alternative input method that benefits individuals with mobility impairments or disabilities, enabling them to interact with digital devices more effectively.

d. **Versatility:** Can be customized and adapted for various applications and industries, including gaming, education, healthcare, and productivity, enhancing versatility and usability across different domains.

13.5 Disadvantages:

a. **Learning Curve:** Users may require time to become familiar with gesture-based controls, potentially leading to a learning curve and initial frustration.

b. **Accuracy Issues:** Gesture recognition systems may encounter accuracy issues, misinterpreting or failing to recognize certain hand gestures accurately, leading to unintended actions or commands.

c. **Fatigue:** Prolonged use of gesture-based controls may result in hand fatigue or discomfort, especially if users are required to hold their arms in specific positions for extended periods.

d. **Limited Gesture Vocabulary:** Gesture-controlled systems may have a limited vocabulary of recognized gestures, restricting the range of commands and interactions available to users.

13.6 Future Scope:

The future of gesture-controlled virtual mouse technology shows great promise, with various opportunities for advancement and application in diverse fields. Gesture-controlled interfaces offer a more intuitive and engaging interaction experience compared to traditional input methods. As technology evolves, improvements in gesture recognition algorithms and will further enhance the user experience, making interactions smoother and more immersive.

Gesture-controlled virtual mouse systems can be enabling new possibilities for education, and productivity applications. This integration allows users to interact with digital content in three-dimensional space using intuitive hand gestures. Gesture-controlled interfaces can improve accessibility and inclusivity by providing alternative input methods for individuals with disabilities or mobility impairments.

As these technologies become more accessible and affordable, they can empower a wider range of users to access and interact with digital devices and applications. Furthermore, gesture-controlled virtual mouse applications can play a crucial role in enhancing productivity and efficiency in various industries and domains.

By enabling hands-free interaction with digital tools and software applications, these applications can streamline workflow processes, improve collaboration, and increase overall productivity. For instance, in office environments, users can perform tasks such as navigating through documents, managing emails, or giving presentations without the need to physically touch a keyboard or mouse.

Moreover, gesture-controlled virtual mouse applications have the potential to revolutionize accessibility for individuals with disabilities. By providing an alternative input method that does not rely on fine motor skills or physical dexterity, these applications can empower users with disabilities to navigate and interact with digital content more effectively. This inclusivity ensures that everyone, regardless of their abilities, can fully participate in the digital world.

13.7 User Manual:

- **Installation:**

i. First download the GCVM.exe Zip File
ii. Then Extract the file in local disk.
iii. Create shortcut of GCVM application to show window.
iv. Open the GCVM application
v. And then showing window & you click on Enable Virtual Mouse.
vi. And then access the camera and ready for use.

- **Usage:**

i. Show hand on your computer/Laptop camera.
ii. And then show v gesture of your camera for Moving Mouse Cursor
iii. Then try to left click and right click using index finger and middle finger respectively.
iv. Using a scrolling event for show pinch gesture in front of camera
v. To use select all item use a fist gesture
vi. To drag and drop event use a two fingers are bend index finger and middle finger.

- **Note:**

ii. Your laptop or Computer is needed for the camera work in this application.
iii. This application can be work only with <u>Right Hand</u>

14. Biography

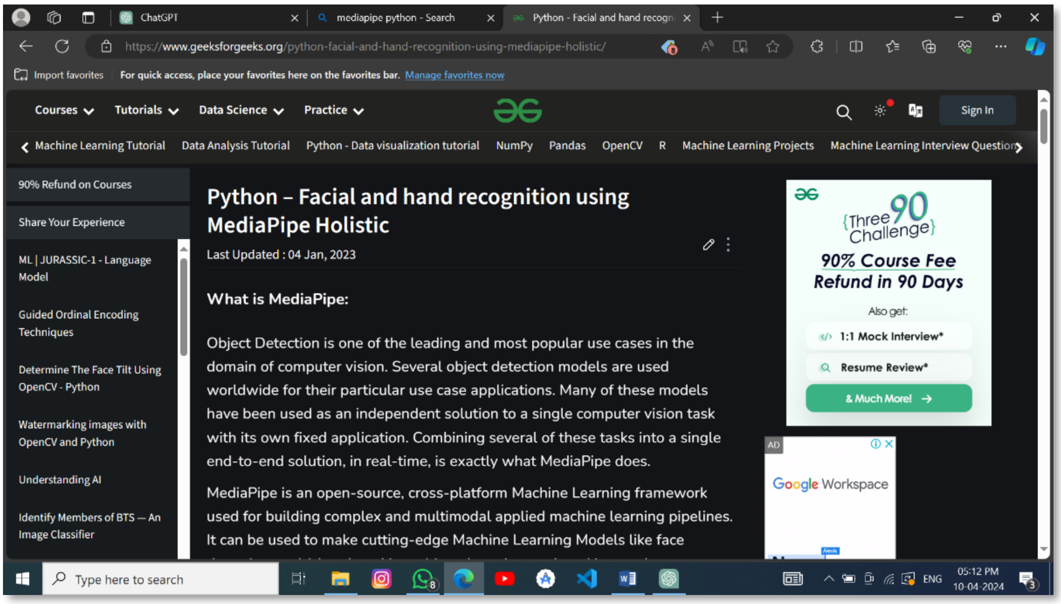
14.1 Book Reference:

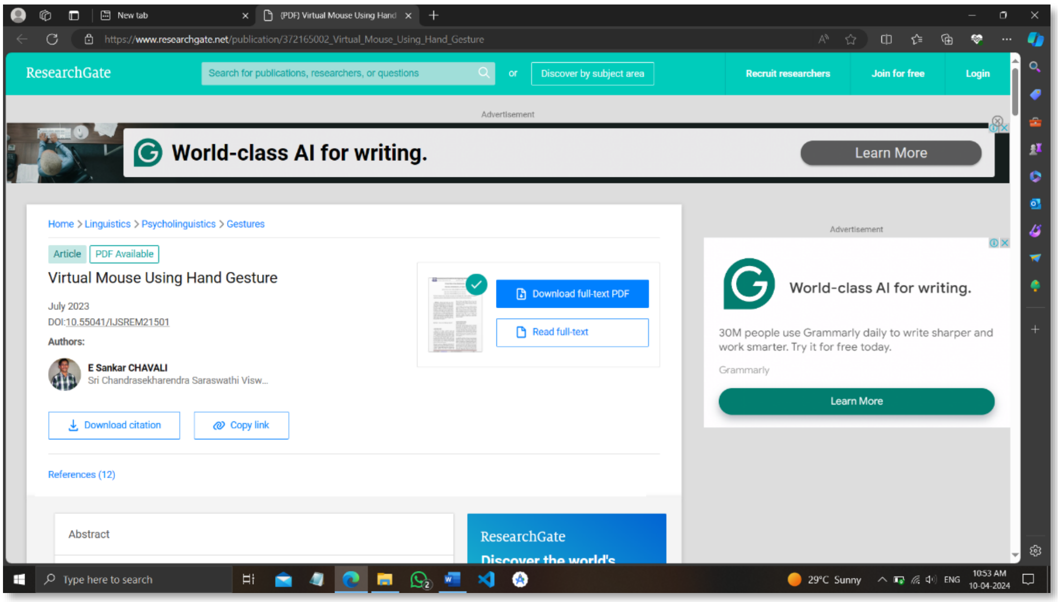
Book Name	Author	Publication	Published year
Python Programming	Rao, K. Nageswara	SciTech Publications PVT. Ltd.	2021
Head First Python, 2nd Edition	Paul, Barry	O 'Reilly Publication	2016
Machine Learning using Python	Manaranjan Pradhan	Wiley	2019

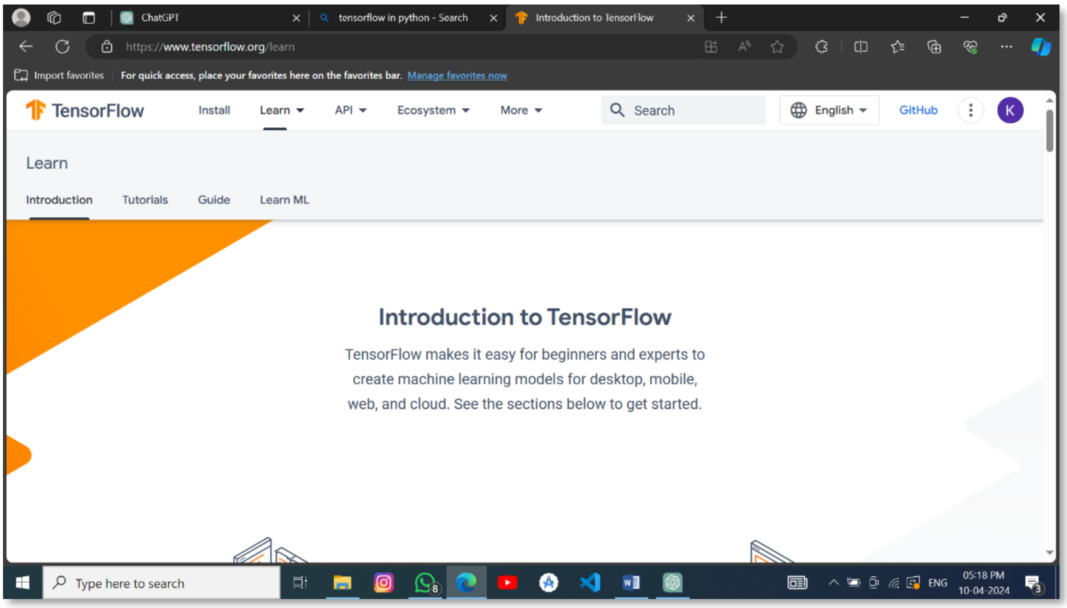
14.2 Web Reference:

Website Name	Website Links
MediaPipe	Python - Facial and hand recognition using MediaPipe Holistic - GeeksforGeeks
OpenCV	Getting Started with OpenCV | LearnOpenCV
Tenserflow	Introduction to TensorFlow

14.3 Screenshots:






Introduction to TensorFlow
TensorFlow makes it easy for beginners and experts to create machine learning models for desktop, mobile, web, and cloud. See the sections below to get started.
TensorFlow
Install
Learn
API
Ecosystem
More
Search
English
GitHub
Learn
Introduction
Tutorials
Guide
Learn ML
https://www.tensorflow.org/learn

Appendix-B

PROGRESSIV ASSESSMENT (PA) OF CAPSTONE PROJECT-EXECUTION AND REPORT WRITING

Evaluation Sheet for Internal Assessment

Name of Student: Karmalkar K. G., Gaikwad R. G., Kulkarni P. A.

Name of Program: Computer Engineering **Semester:** Sixth

Course Title: Capstone Project Execution and Report Writing **Code:** 22060

Title of the capstone project: Gesture Controlled Virtual Mouse

A. Po's addressed by the Capstone Project

a. To apply knowledge of programming language fundamentals and an engineering specialization to the solution for Gesture Control Virtual Mouse.
b. Apply relevant Computer technologies and tools with an understanding of the limitations.

B. Cos addressed by the Capstone Project

a. Implement the planned activity individually or as team.
b. Select, collect and use required information to solve the identified problems.
c. Communicate effectively and confidently as a member and leader of team.

C. Other learning outcomes achieved through this Project

a. Unit outcomes (Cognitive Domain)

1) Apply relevant computer technologies and tools with an understanding of the limitations.
2) Communicate effectively in oral and written form.

b. Practical outcomes (in Psychomotor Domain)

a. Computer Software and Hardware Usage: Use state-of-the-art technologies for operation and application of computer software and hardware.
b. Computer Engineering Maintenance: Maintain computer engineering related software and hardware systems.

c. Affective Domain Outcomes

1) Follow safety measure.

2) Practice good housekeeping.

3) Work as a leader/a team member.