Gesture Controlled Virtual Mouse for Operating laptops/PC’s

# A PROJECT REPORT

### Submitted by

|  |  |
| --- | --- |
| GUNAPU BHARGAVA SAI VARDHAN | 201801330007 |
| KATAKAM SANDEEP | 201801330025 |
| KILLAMSETTY VINEETH | 201801330019 |
| SYED ABDUL REHAN | 201801260007 |
| KALISETTI NIHANTH NAIDU | 201801330008 |
| THUMATI HEMANTH | 201801330024 |

***Under the esteemed guidance of***

**Mr. R LAKSHMAN M. Tech, (Ph. D), Assistant Professor**

**in partial fulfilment for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING SCHOOL OF ENGINEERING AND TECHNOLOGY

**VIZIANAGARAM CAMPUS**

**CENTURION UNIVERSITY OF TECHNOLOGY AND MANAGEMENT**

# ANDHRA PRADESH DECEMBER 2023 / MAY 2024

**BONAFIDE CERTIFICATE**

## Certified that this project report Developing a solution for Gesture enabled commands for operating Laptops/PCs for frequently used operations on a daily basis. The bonafide work of “GUNAPU BHARGAV SAI VARDHAN (201801330007), KATAKAM SANDEEP (201801330025), KILLAMSETTY VINEETH (201801330019), SYED ABDUL REHAN(201801260007), KALISETTI NIHANTH NAIDU (201801330008), THUMATI. HEMANTH (201801330024)” who carried out the project work under my supervision. This is to further certify to the best of my knowledge that this project has not been carried out earlier in this institute and the university.

SIGNATURE Mr. R. LAKSHMAN

ASSISTANT PROFESSOR

*Certified that the above-mentioned project has been duly carried out as per the norms of the college and statutes of the university.*

SIGNATURE

Dr. P. SUBRAT KUMAR

HOD, Associate Professor

SIGNATURE

Dr. P.A. SUNNY DAYAL

DEAN, Associate Professor

## HEAD OF THE DEPARTMENT / DEAN OF THE SCHOOL

**Professor of Computer Science and Engineering DEPARTMENT SEAL**

# DECLARATION

We hereby declare that the project entitled **“Developing a solution for Gesture enabled commands for operating Laptops/PCs for frequently used operations on a daily basis.”** submitted to the fulfilment of the award of the degree of B.Tech(Cse) in Centurion University of Technology and Management, Vizianagaram. This project work in original has not been submitted so far in any part or full for any other university or institute for the award of any degree or diploma.

|  |  |
| --- | --- |
| GUNAPU BHARGAVA SAI VARDHAN | 201801330007 |
| KATAKAM SANDEEP | 201801330025 |
| KILLAMSETTY VINEETH | 201801330019 |
| SYED ABDUL REHAN | 201801260007 |
| KALISETTI NIHANTH NAIDU | 201801330008 |
| THUMATI HEMANTH | 201801330024 |

# ACKNOWLEDGEMENTS

I am immensely thankful to Assistant Professor P. Anuradha, of the Department of Computer Science and Engineering at SoET, Vizianagaram Campus. P. Anuradha Ma’am led me through the complexities of this project effortlessly, displaying unparalleled generosity and guidance.

I wish to express my profound and sincere gratitude to Associate Professor Dr. N. V. S. Shankar, Department of Computer Science and Engineering, SoET, Vizianagaram Campus, who guided me into the intricacies of this project nonchalantly with matchless magnanimity.

I thank Prof. Dr. Subrat Kumar Parida, Head of the Dept. of Department of Computer Science and Engineering, SoET, Vizianagaram Campus for extending their support during Course of this investigation.

I thank Dr. P. A. Sunny Dayal, Dean of SoET, Vizianagaram Campus for their invaluable guidance, insightful feedback, and continuous support throughout the course of this project. Your expertise and mentorship have been invaluable.

I thank Dr. P. Pallavi, Registrar, CUTM, Vizianagaram Campus for their assistance and cooperation in facilitating the necessary resources and administrative support essential for the successful execution of this project.

I thank P. Prasanta Kumar Mohanty, Vice Chancellor, CUTM, Vizianagaram Campus for fostering an environment that encourages academic excellence and innovation. Your vision has been a constant source of inspiration.

I also express my deepest appreciation to my parents for their unconditional love, encouragement, and belief in my abilities. Their unwavering support has been the cornerstone of my achievements.

I am sincerely grateful to each one of you for your contributions, guidance, and unwavering support, without which this project would not have been possible.

# ABSTRACT

Now a days computer vision has reached its pinnacle, where a computer can identify its owner using a simple program of image processing. In this stage of development, people are using this vision in many aspects of day-to-day life, like Face Recognition, Color detection, Automatic car, etc. In this project, computer vision is used in creating an Optical mouse and keyboard using hand gestures. The camera of the computer will read the image of different gestures performed by a person’s hand and according to the movement of the gestures the Mouse or the cursor of the computer will move, even perform right and left clicks using different gestures. Similarly, the keyboard functions may be used with some different gestures, like using one finger gesture for alphabet select and four-figure gesture to swipe left and right. It will act as a virtual mouse and keyboard with no wire or external devices. The only hardware aspect of the project is a web-cam and the coding is done on python using Anaconda platform. Here the Convex hull defects are first generated and then using the defect calculations an algorithm is generated and mapping the mouse and keyboard functions with the defects. Mapping a couple of them with the mouse and keyboard, the computer will understand the gesture shown by the user and act accordingly.

**Keywords—**Gesture control, Media pipe, OpenCV.

# INDEX

Table of Contents

[ACKNOWLEDGEMENTS i](#_Toc162683923)

[ABSTRACT 2](#_Toc162683924)

[INDEX 3](#_Toc162683925)

[LIST OF FIGURES 5](#_Toc162683926)

[CHAPTER 1: INTRODUCTION 5](#_Toc162683927)

[1.1 Purpose 5](#_Toc162683928)

[1.2 Intended Audience 2](#_Toc162683929)

[1.3 Scope 2](#_Toc162683930)

[CHAPTER 2: LITERATURE SURVEY 3](#_Toc162683931)

[CHAPTER 3: OVERALL DESCRIPTION 7](#_Toc162683932)

[3.1 Existing system 7](#_Toc162683933)

[3.2 Proposed System: 7](#_Toc162683934)

[User Characteristics: 10](#_Toc162683935)

[3.1.1 System Requirements Specification: 10](#_Toc162683936)

[3.1.2 Functional Requirements: 11](#_Toc162683937)

[3.1.3 Non- Functional Requirements: 12](#_Toc162683938)

[3.1.4 Hardware Requirements: 13](#_Toc162683939)

[3.1.5 Software Requirements: 14](#_Toc162683940)

[**3.2 UML Diagrams 15**](#_Toc162683941)

[3.2.1 Usecase Diagram 18](#_Toc162683942)

[3.2.2 Sequence Diagram 21](#_Toc162683943)

[3.2.3 Flow Chart 23](#_Toc162683944)

[CHAPTER 4: LIBRARIES 24](#_Toc162683945)

[4.1 LIBRARIES USED 24](#_Toc162683946)

[CHAPTER 5: RESULT AND DISCUSSION 27](#_Toc162683947)

[CHAPTER 6: IMPLEMENTATION 29](#_Toc162683948)

[CONCLUSION 32](#_Toc162683949)

[REFERENCES 33](#_Toc162683950)

# LIST OF FIGURES

FIG 1: IMPLEMENTATION 1

FIG 2: UML DIAGRAM 17

FIG 3: USE CASE DIAGRAM 20

FIG 4: SHOWING SEQUENTIAL DIAGRAM 22

FIG 5: FLOWCHART 23

FIG 6: IMPLEMENTATION 30

FIG 7: OUTPUT IMPLEMENTATION 32

FIG 8 : VOICE ASSISTANT 46

# CHAPTER 1: INTRODUCTION

The Computer webcam is capturing the video of the person sitting in front of the computer, there will be a small green box which will be generated in the middle of the screen. In that green box, the objects shown will be processed by the code and matched with it if it matches then a red colored border will be generated, which means the computer has identified the object and then by moving the object the mouse cursor can be moved. This will not only help in the security of the computer but also help in generating a virtual computational experience. Here in the place of different objects, using hand gestures one gesture will be moving the cursor, the different gesture will be used for right click and different for left click, similarly with a simple gesture can do the keyboard functions virtually that may have been done on some keyboard as a physical aspect. The field of Human-Computer Interaction has seen significant advancements with the introduction of innovative technologies. Traditional input methods such as keyboards, mice, and touchscreens have become more sophisticated, but still require direct contact with the computer, limiting the scope of interaction. Gesture-based interaction has emerged as an alternative approach to traditional methods, and the Gesture Controlled Virtual Mouse is an innovative technology that enables intuitive interaction between humans and computers. This research paper presents a comprehensive study of the Gesture Controlled Virtual Mouse, which leverages state of-the-art Machine Learning and Computer Vision algorithms to enable users to control input/output operations using hand gestures and voice commands without the need for direct contact. The Gesture Controlled Virtual Mouse is designed using the latest technology and is capable of recognizing both static and dynamic hand gestures in addition to voice commands, making the interaction more natural and user friendly. The system does not require any additional hardware, and the implementation of the system is based on models such as the Convolutional Neural Network (CNN) implemented by Media Pipe running on top of pybind11. The system comprises two modules, one of which operates directly on hands using Media Pipe hand detection, while the other module uses gloves of any uniform color. The system currently supports the Windows platform

## Purpose

The purpose of this project is to develop a gesture-controlled virtual mouse system that

enables users to operate laptops/PCs using hand gestures instead of conventional input

devices like a physical mouse or touchpad. By leveraging motion sensing technology,

such as accelerometers or cameras, the system aims to provide an intuitive and

immersive user experience, enhancing accessibility and convenience for individuals

with mobility impairments or for scenarios where traditional input methods are not

feasible or convenient.

## Intended Audience

The intended audience for this project includes:

* Individuals interested in human-computer interaction (HCI) and novel input methods.
* Developers and researchers in the fields of computer vision, machine learning, and

sensor technology.

* Users with physical disabilities or limited mobility seeking alternative input methods for operating computers.
* Educational institutions and students studying computer science, electrical engineering,

Or Related Disciplines

## Scope

The scope of this project encompasses the following key components and functionalities:

**Gesture Recognition:**

Implementing algorithms for real-time detection and recognition of hand gestures using sensors or cameras.

**Virtual Mouse Control:** Developing software to translate recognized gestures into mouse cursor movements and actions (e.g., clicks, scrolls).

**Integration with Operating Systems:** Ensuring compatibility with popular operating systems (e.g., Windows, macOS, Linux) to enable seamless interaction with a wide range of laptops/PCs.

**User Interface:** Designing a user-friendly interface for configuring gesture settings, calibrating sensors, and providing feedback to users.

**Performance Optimization:** Optimizing the system for low latency and high accuracy to deliver a responsive and reliable user experience.

**Testing and Evaluation:** Conducting comprehensive testing and evaluation to assess the system's usability, effectiveness, and potential limitations in various usage scenarios.

**Documentation and Reporting:** Documenting the development process, technical specifications, and experimental results in a project report to disseminate knowledge and facilitate further research and development in the field.

# CHAPTER 2: LITERATURE SURVEY

Gesture-based mouse control using computer vision has been a topic of interest for researchers for a long time. Various methods have been proposed for gesture recognition, but in this paper, the authors have proposed a new method based on color detection and masking. This system is implemented in Python programming language using the OpenCV library, which is a popular computer vision library. The proposed system is a virtual mouse that will work only based on webcam captured frames and tracking colored fingertips.

The objective of this paper is to develop and implement an alternative system to control a mouse cursor. The alternative method is hand gesture recognition using a webcam and a color detection method. The ultimate outcome of this paper is to develop a system that recognizes hand gestures and controls the mouse cursor using the color detection method of any computer. The system works on the frames captured by the webcam on the computer machine or built-in camera on a laptop. By creating the video capture object, the system will capture video using the webcam in real-time. The camera should be positioned in a way so that it can see the user’s hands in the right positions. In the previously proposed system by Kabid Hassan Shibly’s "Design and Development of Hand Gesture Based Virtual Mouse" research paper published in ICASERT (2019), color detection is done by detecting color pixels of fingertips with color caps from the frames that were captured by the webcam. This is the initial and funda- mental step of the proposed system. The outcome of this step will be a grayscale image, where the intensity of the pixels differs from the color cap to the rest of the frame, and the color cap area will be highlighted. Then, rectangle bounding boxes (masks) will be created around the color cap, and the color cap will be tracked. The gesture will be detected from the tracking of these color caps.At first, the center of two detected color objects is calculated, which is done by the coordinates of the center of the detected rectangle. To create a line between two coordinates, the built-in OpenCV function is used, and to detect the midpoint equation, a given formula is used. This midpoint is the tracker for the mouse pointer, and the mouse pointer will track this midpoint. In this system, the coordinates from camera captured frames resolution are converted to screen resolution. A predefined location for the mouse is set, so that when the mouse pointer reaches that position, the mouse started to work, and this may be called an open gesture. This allows the user to control the mouse pointer. The previous system uses close gestures for clicking events. When the rectangle bounding boxes come closer to another rectangle, the bounding box is created with the edge of the tracking bounding boxes. When the newly created bounding box becomes 20 percent of its creation time size, the system performs the left button click, and it can be clicked. By holding this position more than 5 seconds, the user can perform a double-click. And for the right button click, again the open gesture is used. To perform the right button click, a single finger is good enough. The system will detect one fingertip color cap, then it performs a right button click. To scroll with this system, the user needs to use the open gesture movement with three fingers with color caps. If the users use their three fingers together and change its position to downwards, it will perform scrolling down. Similarly, if its position is changed to upwards, it will perform scrolling up. When three fingers move up or down, the color caps get a new position and new coordinates. By the time all three-color caps get new coordinates, it performs scrolls. If their y coordinate values decrease, it will perform scrolling down, and if the values increase, it will perform scrolling up. In conclusion, the proposed system has shown a new method for gesture-based mouse control using computer vision. The system uses color detection and masking to recognize hand gestures and control the mouse cursor.

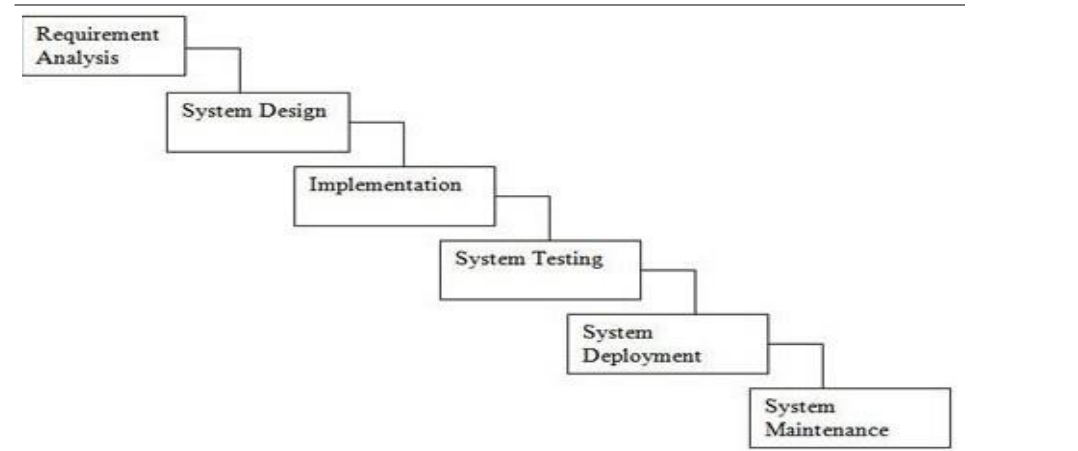


FIG 1:IMPLEMENTATION

1. **Introduction:**

Gesture control interfaces have emerged as an alternative and potentially more intuitive way of interacting with computers and electronic devices. Traditional input methods like keyboards and mice have limitations in terms of ease of use and accessibility, especially for individuals with physical disabilities. Gesture control systems offer a promising solution by allowing users to interact with devices through natural hand movements and gestures.

Advancements in sensor technologies, such as depth sensors, cameras, accelerometers, and gyroscopes, coupled with sophisticated machine learning algorithms, have propelled the development of gesture-controlled systems. These systems can recognize and interpret various hand movements, enabling users to navigate interfaces, control applications, and perform actions without the need for physical touch or input devices.

**2. Gesture Recognition Techniques:**

a. **Computer Vision-based Approaches:** Computer vision techniques analyze images or 3D data captured by cameras or depth sensors to recognize hand movements and gestures. Background subtraction is used to isolate the hand from the background, while hand tracking algorithms track the movement of the hand in real-time. Feature extraction methods identify distinctive features of the hand, such as finger positions and palm orientation, for gesture recognition.

Sensors like accelerometers and gyroscopes measure the motion of the hand and capture motion data, which is then processed using machine learning algorithms. Support Vector Machines (SVM) or Neural Networks are commonly used to classify motion patterns and recognize gestures based on sensor data.

c. **Hybrid Approaches:** Hybrid approaches combine multiple sensing modalities, such as vision and inertial sensors, to improve gesture recognition accuracy and robustness. By fusing data from different sensors, these hybrid systems can mitigate the limitations of individual sensing technologies and provide more reliable gesture recognition.

**3. Virtual Mouse Control:**

a. **Direct Mapping:** In direct mapping, simple gestures are directly mapped to specific mouse actions. For example, moving the hand left/right can control cursor movement, and making a fist can simulate a mouse click. While intuitive, direct mapping approaches may lack flexibility and may not support complex interactions.

**b. Gesture Sequences:** Gesture sequences involve interpreting sequences of gestures or hand movements as multi-step commands. For example, a swipe followed by a tap could simulate a drag-and-drop operation or a two-finger pinch gesture could simulate zooming in or out.

c. **Dynamic Gestures:** Dynamic gestures consider the speed, trajectory, or acceleration of hand movements to influence the corresponding mouse actions. For example, a faster hand movement might result in faster cursor movement, while a slower movement might result in more precise control.

**4. Applications and Use Cases:**

Gesture-controlled virtual mouse systems have diverse applications and use cases, including:

a**. Accessibility:** These systems provide alternative input methods for individuals with mobility impairments or disabilities, allowing them to interact with laptops/PCs more comfortably and efficiently.

b. **Gaming:** Gesture-controlled systems enhance gaming experiences by enabling gesture-based control schemes for navigating game menus, controlling characters, or performing in-game actions.

c**. Presentations:** Presenters can use gesture-controlled systems to navigate slides and interact with content during presentations, eliminating the need for traditional input devices and enhancing engagement with the audience.

d. **Virtual Reality (VR) and Augmented Reality (AR):**

Gesture control interfaces integrated with VR/AR systems enable natural interaction with virtual environments and virtual objects, enhancing immersion and user experience.

**5. Challenges and Future Directions:**

Gesture-controlled virtual mouse systems face several challenges and opportunities for future research and development, including:

a**. Accuracy and Robustness:** Improving the accuracy and robustness of gesture recognition algorithms, especially in challenging environments with varying lighting conditions, backgrounds, and noise levels.

b. **User Experience:** Enhancing the user experience by refining gesture control interfaces, reducing latency, and optimizing interaction design for intuitive and effortless operation.

c**. Adaptability:** Designing systems that can adapt to individual user preferences and accommodate diverse hand sizes, shapes, and gestures to improve usability and accessibility.

d**. Privacy and Security:** Addressing privacy concerns related to the use of cameras or sensors for gesture recognition and ensuring the security of user data in gesture-controlled systems through encryption and secure data handling practices.

**6.Conclusion:**

Gesture-controlled virtual mouse systems represent a promising paradigm for human-computer interaction, offering novel ways of operating laptops/PCs and other electronic devices. By leveraging advances in sensor technologies, machine learning algorithms, and interface design, researchers and developers continue to explore innovative approaches to enhance the usability, accessibility, and user experience of gesture-controlled systems.

This detailed elaboration provides a deeper understanding of each topic covered in the literature survey, highlighting key concepts, techniques, applications, challenges, and future directions in the field of gesture-controlled virtual mouse systems for operating laptops/PCs.

# CHAPTER 3: OVERALL DESCRIPTION

The gesture-controlled virtual mouse system is designed to provide users with an intuitive and natural way of interacting with laptops/PCs by using hand gestures to control the cursor and perform mouse actions. The system comprises hardware components for capturing hand movements and gestures, software components for gesture recognition and virtual mouse control, and user interface components for configuration and feedback.

## Existing system

The existing system typically consists of the following components:

**3.1**.**1Hardware Components:**

Cameras or depth sensors: These capture images or 3D data of hand movements for gesture recognition.

Inertial sensors (e.g., accelerometers, gyroscopes): These measure motion data from hand movements for gesture recognition.

**3.1.2 Software Components:**

Gesture Recognition Algorithms: These analyse data from sensors to recognize hand movements and gestures.

Virtual Mouse Control Software: This translates recognized gestures into corresponding mouse actions, such as cursor movements, clicks, and scrolls.

Integration with Operating System: The software interacts with the operating system to control the cursor and perform mouse actions.

**3.1.3 User Interface:**

Configuration Interface: Users can configure gesture settings, calibrate sensors, and customize gesture commands through a user-friendly interface.

Feedback Mechanism: The system provides visual or auditory feedback to users to indicate successful gesture recognition and virtual mouse control.

## 3.2 Proposed System:

The proposed system aims to enhance the existing system by incorporating the following improvements and additional features:

**3.2.1 Hardware Upgrades:**

Enhanced Sensor Technology: Upgrading cameras or depth sensors to improve accuracy and robustness of gesture recognition.

Multi-Sensor Fusion: Integrating multiple sensing modalities (e.g., vision and inertial sensors) for improved gesture recognition accuracy and reliability.

**3.2.2 Software Enhancements:**

**Advanced Gesture Recognition Algorithms:**

**Convolutional Neural Networks (CNNs):** CNNs are deep learning models widely used in computer vision tasks, including gesture recognition. These networks can automatically learn hierarchical features from image data, making them well-suited for analyzing hand gestures captured by cameras or depth sensors. CNN-based gesture recognition models can achieve high accuracy and robustness, especially when trained on large and diverse datasets.

**Recurrent Neural Networks (RNNs):** RNNs are another class of deep learning models commonly used for sequential data processing, making them suitable for analyzing dynamic hand gestures captured by inertial sensors. By leveraging sequential information, RNNs can capture temporal dependencies in gesture sequences, enabling more accurate recognition of complex gestures with dynamic characteristics.

**Hybrid Models:** Hybrid models that combine CNNs and RNNs can leverage the strengths of both architectures for gesture recognition. For example, a hybrid model may use a CNN to analyze images of hand gestures captured by cameras and an RNN to process motion data from inertial sensors, leading to more comprehensive and robust gesture recognition.

**Dynamic Gesture Analysis:**

**Speed, Trajectory, and Acceleration Analysis:** Implementing algorithms to analyze dynamic aspects of gestures, such as speed, trajectory, and acceleration, can provide finer control over virtual mouse movements. By incorporating dynamic gesture analysis, the system can differentiate between subtle variations in hand movements and translate them into corresponding mouse actions with varying speed and precision.

**Gesture Segmentation:** Segmenting continuous hand movements into discrete gestures enables the system to recognize and interpret individual gestures accurately. Techniques such as peak detection or dynamic time warping can be employed to identify key points or segments in gesture sequences, facilitating gesture recognition and virtual mouse control.

**Gesture Prediction:**

**Sequential Modeling:** Implementing sequential modeling techniques, such as Markov models or Long Short-Term Memory (LSTM) networks, to predict future gestures based on previous interactions. By analyzing temporal patterns in gesture sequences, the system can anticipate user gestures and initiate virtual mouse actions proactively, enhancing responsiveness and user experience.

**Contextual Prediction:** Considering contextual information, such as the current application or user activity, to improve gesture prediction accuracy. For example, the system can learn context-specific gesture patterns for common tasks like scrolling in web browsers or navigating menus in applications, enabling more accurate prediction of user intentions.

**Optimization Techniques:**

**Model Compression:** Compressing gesture recognition models to reduce memory footprint and computational overhead, making them suitable for deployment on resource-constrained devices such as smartphones or embedded systems. Techniques like quantization, pruning, and knowledge distillation can be applied to compress deep learning models without significantly compromising performance.

**Real-time Performance Optimization:** Implementing optimization techniques to improve real-time performance and responsiveness of gesture recognition algorithms. Techniques such as parallelization, vectorization, and hardware acceleration (e.g., GPU acceleration) can be utilized to optimize computation and reduce latency, ensuring smooth and responsive interaction with the virtual mouse system.

**Integration with Operating System:**

**Device Driver Integration:** Developing device drivers or software libraries to facilitate seamless integration of the gesture-controlled virtual mouse system with the underlying operating system. These drivers enable the system to communicate with the operating system's input subsystem and emulate mouse actions effectively, ensuring compatibility and interoperability with a wide range of platforms and applications.

**Gesture Mapping Configuration:** Providing user-friendly interfaces or configuration tools to customize gesture mappings and define user-specific gesture commands. Users can map specific hand gestures to desired mouse actions (e.g., cursor movement, clicks, scrolls) according to their preferences and usage scenarios, enhancing flexibility and usability.

**3.2.3 User Interface Improvements:**

**3.2.3.1Gesture Visualization:**

**Real-time Feedback:** Provide visual feedback to users in real-time to indicate recognized gestures. This can include highlighting the detected hand movements or displaying graphical representations of recognized gestures on the screen.

**Gesture Trail:** Display a trail or path representing the trajectory of hand movements to help users visualize their gestures and understand how they are interpreted by the system.

**Confidence Level:** Display the confidence level or certainty of gesture recognition to inform users about the reliability of the system's interpretation of their gestures.

**3.2.4 Performance Optimization:**

**Efficient Gesture Recognition Algorithms:** Implement optimized algorithms for gesture recognition that strike a balance between accuracy and computational efficiency.

**Parallelization:** Utilize parallel processing techniques to distribute computational tasks across multiple processing units (e.g., CPU cores, GPU cores) for faster execution.

**3.2.4.1 Power Efficiency:**

**Power-aware Design:** Design the gesture-controlled virtual mouse system with power efficiency in mind to minimize energy consumption and maximize battery life, especially for mobile devices.

**Low-Power Modes:** Implement low-power modes or sleep modes that automatically activate during periods of inactivity to conserve energy.

## User Characteristics:

Understanding user characteristics is crucial for designing an effective and user-friendly gesture- controlled virtual mouse system. Here are some key user characteristics to consider:

**Physical Abilities:**

**Fine Motor Skills:** Consider the user's fine motor skills, including their ability to perform precise hand movements and gestures. Users with limited fine motor skills may require larger and more easily recognizable gestures for effective interaction.

**Cognitive Abilities:**

Consider the cognitive load imposed by gesture commands and interactions. Users with cognitive impairments or limited attention span may benefit from simple and intuitive gestures that are easy to remember and execute.

## System Requirements Specification:

Software Requirements Specification (SRS) – a requirements specification for a software system is a complete description of the behaviour of a system to be developed. It includes a set of cases that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. Non-functional requirements are requirements that impose constraints on the design or implementation such as performing engineering requirements, quality standards, or design constraints.

## Functional Requirements:

**Gesture Recognition**:

The system should accurately recognize a predefined set of hand gestures as input commands.

It should support various types of gestures, including single-hand gestures, multi-hand gestures, and dynamic gestures.

The system should be capable of distinguishing between different gestures and mapping them to specific actions.

**Virtual Mouse Control:**

The system should accurately translate recognized gestures into corresponding mouse actions, such as cursor movement, clicks, and scrolls.

It should provide smooth and responsive control of the virtual mouse, with minimal latency between gesture input and mouse response.

The system should support a range of mouse actions, including left-click, right-click, double-click, drag-and-drop, and scrolling.

**Customization and Configuration:**

Users should be able to customize and configure gesture mappings, defining their own set of gestures and associating them with specific mouse actions.

The system should provide a user-friendly interface for configuring gesture settings, calibrating sensors, and adjusting sensitivity levels.

It should support saving and loading of user-defined configurations for easy customization and portability across devices.

**Compatibility and Integration:**

The system should be compatible with a variety of hardware devices, including cameras, depth sensors, accelerometers, and gyroscopes.

It should integrate seamlessly with popular operating systems (e.g., Windows, macOS, Linux) and applications, allowing users to interact with their computers using gesture controls in a wide range of contexts.

The system should support integration with existing input devices (e.g., keyboards, touchpads) for hybrid input scenarios and fallback options.

**Feedback and Visualization:**

The system should provide visual feedback to users to indicate recognized gestures and virtual mouse actions in real-time.

It should include graphical indicators or animations to visualize the trajectory of hand movements and the position of the virtual cursor.

The system should provide auditory feedback options for users who may benefit from additional cues or alerts during interaction.

**Accessibility and Usability:**

The system should be accessible to users with diverse abilities, including those with physical disabilities or limited fine motor skills.

It should include accessibility features such as alternative input modalities, customizable gesture sensitivity, and options for adjusting interaction speed.

The system should prioritize usability and user experience, with intuitive interactions, clear feedback mechanisms, and minimal cognitive load.

**Performance and Optimization:**

The system should be optimized for real-time performance, with minimal latency and smooth responsiveness during gesture recognition and virtual mouse control.

It should implement efficient algorithms and data processing techniques to minimize computational overhead and maximize efficiency, especially on resource-constrained devices.

The system should optimize power consumption to ensure energy efficiency and maximize battery life, particularly for mobile devices.

## Non- Functional Requirements:

Non-functional requirements define the attributes that describe the quality of the system and its characteristics rather than specific behaviors. Here are some non-functional requirements for your gesture-controlled virtual mouse project:

**Performance:**

**Response Time:** The system should respond to recognized gestures and translate them into corresponding mouse actions within milliseconds to provide a smooth and seamless user experience.

**Reliability:**

**Accuracy:** The system should accurately recognize and interpret hand gestures with a high level of precision to minimize errors and false positives during virtual mouse control.

**Security:**

**Data Privacy:** The system should prioritize user privacy and data protection by implementing secure data handling practices, encryption mechanisms, and access control measures to safeguard sensitive user information collected during gesture recognition.

**Usability:**

**User Interface Design:** The user interface should be intuitive, visually appealing, and easy to navigate, with clear instructions, visual cues, and feedback mechanisms to guide users in performing gestures and interacting with the virtual mouse.

**Code Maintainability:** The system codebase should adhere to coding standards, documentation guidelines, and version control best practices to ensure readability.

## Hardware Requirements:

**3.1.4.1 Sensor Hardware**:

**Camera or Depth Sensor:** Depending on your chosen approach, you'll need a camera or depth sensor to capture hand movements and gestures. This could be a standard RGB camera, infrared camera, or depth sensor such as Kinect or Intel RealSense.

**Inertial Sensors:** For additional input data and improved accuracy, consider integrating inertial sensors like accelerometers and gyroscopes. These sensors can capture motion data and supplement the information obtained from cameras or depth sensors.

**3.1.4.2 Processing Hardware:**

**CPU:** A powerful CPU is essential for real-time processing of sensor data, gesture recognition algorithms, and virtual mouse control. Multi-core processors with high clock speeds are preferable for efficient computation.

**GPU (Optional):** Depending on the complexity of your gesture recognition algorithms and virtual mouse control, a GPU may be beneficial for parallel processing and acceleration of certain tasks, especially deep learning inference.

**RAM:** Sufficient RAM is necessary for storing sensor data, intermediate results, and running gesture recognition algorithms. The amount of RAM required will depend on the size of data being processed and the complexity of algorithms.

**3.1.4.3 Storage:**

**Internal Storage:** Adequate internal storage is needed for storing system software, configuration files, and any data generated or logged by the system. Solid-state drives (SSDs) are preferred for faster data access and system responsiveness.

**External Storage (Optional):** Depending on your requirements, you may need external storage devices for storing large datasets, training models, or additional software packages.

­­

**3.1.4.4Connectivity:**

**USB Ports:** Sufficient USB ports are required for connecting cameras, depth sensors, inertial sensors, and other peripheral devices to the system. Consider the number and type of USB ports available on your hardware platform.

**Wireless Connectivity (Optional):** Wi-Fi or Bluetooth connectivity may be required for wireless communication with external devices, remote control, or data transfer.

**3.1.4.5Display and Output Devices:**

**Monitor or Display:** A monitor or display is necessary for visualizing the virtual mouse cursor and user interface. Consider the resolution, size, and type of display based on your application requirements.

**Input Devices**: While the project focuses on gesture-based interaction, it's important to have traditional input devices like keyboards and mice as backup options, especially during system setup and configuration.

## Software Requirements:

Software requirements for your gesture-controlled virtual mouse project encompass the necessary software components, libraries, and development environments needed to implement the system. Here's an overview of the software requirements:

**Operating System:**

Determine the operating system(s) supported by your gesture-controlled virtual mouse system. Common choices include:

Windows

macOS

Linux distributions (e.g., Ubuntu, Fedora)

Specify any specific versions or editions required for compatibility with your software components.

**Development Environment:**

Choose a development environment for software development and testing:

Integrated Development Environment (IDE): Examples include Visual Studio, JetBrains IntelliJ IDEA, or Eclipse.

**Programming Languages:**

Determine the programming languages to be used for implementing the gesture-controlled virtual mouse system:

**Gesture Recognition Libraries:**

Identify and integrate gesture recognition libraries for analyzing hand movements and recognizing predefined gestures:

OpenCV (OpenSource Computer Vision Library): Widely used for image processing and computer vision tasks.

TensorFlow or PyTorch: Deep learning frameworks for training and deploying gesture recognition models.

Media Pipe: Google's open-source framework for building pipelines for machine learning, especially for hand tracking and gesture recognition.

Open Pose: Library for real-time multi-person keypoint detection and pose estimation.

**Virtual Mouse Control Software:**

Develop software components for translating recognized gestures into corresponding mouse actions:

Implement algorithms for cursor movement, left-click, right-click, scrolling, and other mouse actions.

Integrate with system-level APIs or libraries for emulating mouse input on the operating system.

## UML Diagrams

To create a UML diagram for your gesture-controlled virtual mouse project, we can start by identifying the main components and their relationships within the system. Here's a description of the key components and their interactions:

**User Interface (UI):**

The UI component represents the graphical user interface through which users interact with the gesture-controlled virtual mouse system.

It includes elements such as buttons, sliders, and visual feedback displays for gesture recognition and virtual mouse control.

The UI interacts with the Gesture Recognition and Virtual Mouse Control components to receive input data and provide feedback to the user.

**Gesture Recognition:**

The Gesture Recognition component processes input data from sensors (e.g., cameras, depth sensors, inertial sensors) to recognize hand gestures.

It includes algorithms for hand tracking, gesture detection, and gesture classification using computer vision, machine learning, or a combination of both.

The Gesture Recognition component communicates with the UI component to send recognized gesture data for further processing.

**Virtual Mouse Control:**

The Virtual Mouse Control component translates recognized gestures into corresponding mouse actions, such as cursor movement, clicks, and scrolls.

It emulates mouse input commands to interact with the operating system and applications running on the computer.

The Virtual Mouse Control component interacts with the UI component to receive gesture commands and provide feedback on virtual mouse actions.

**Hardware Interface:**

The Hardware Interface component manages communication with hardware devices, such as cameras, depth sensors, and inertial sensors.

It includes drivers or APIs for accessing sensor data and configuring sensor parameters.

The Hardware Interface component interacts with the Gesture Recognition and Virtual Mouse Control components to provide input data for gesture recognition and virtual mouse control.

**Configuration Management:**

The Configuration Management component handles user-defined configurations for gesture mappings, sensitivity settings, and other system parameters.

It provides interfaces for users to customize gesture settings and save/load configuration profiles.

The Configuration Management component interacts with the UI component to display configuration options and apply user-defined settings.

**Logging and Analytics:**

The Logging and Analytics component records system events, user interactions, and performance metrics for monitoring and analysis purposes.

It collects data on recognized gestures, virtual mouse actions, and system usage patterns.

The Logging and Analytics component may interact with external tools or services for data storage and analysis.

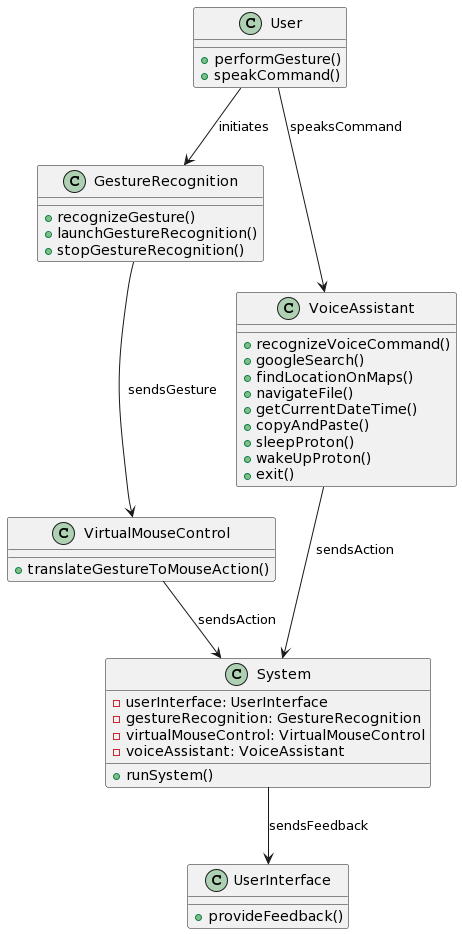


FIG:1 UML DIAGRAM

## Usecase Diagram

A use case diagram provides a high-level view of the interactions between actors (users or external systems) and the system itself. In the context of your gesture-controlled virtual mouse project, here's a description of the main actors and their interactions:

**User:**

The primary actor in the system, representing the end user who interacts with the gesture-controlled virtual mouse system.

**System:** The gesture-controlled virtual mouse system itself, which encompasses the various components responsible for gesture recognition, virtual mouse control, user interface, and hardware integration.

Here's a description of the main use cases in the system:

**Perform Gesture:**

Actors: User

Description: The user performs a hand gesture in front of the sensor device to control the virtual mouse.

Preconditions: The system is active and ready to receive gesture input.

Postconditions: The system recognizes the gesture and translates it into corresponding mouse actions.

**Configure Gesture Settings:**

Actors: User

Description: The user configures gesture settings, including gesture mappings, sensitivity levels, and other system parameters.

Preconditions: The user has access to the system configuration interface.

Postconditions: The user-defined gesture settings are applied, affecting the behavior of the gesture-controlled virtual mouse system.

**Calibrate Sensors:**

Actors: User

Description: The user calibrates sensor devices, such as cameras or depth sensors, to optimize gesture recognition accuracy.

Preconditions: The system supports sensor calibration functionality.

Postconditions: Sensor devices are calibrated, improving the accuracy of gesture recognition.

**Provide Feedback:**

Actors: System

Description: The system provides visual and/or auditory feedback to the user to indicate recognized gestures and virtual mouse actions.

Preconditions: The system successfully recognizes a gesture or performs a virtual mouse action.

Postconditions: The user receives feedback confirming the system's response to their input.

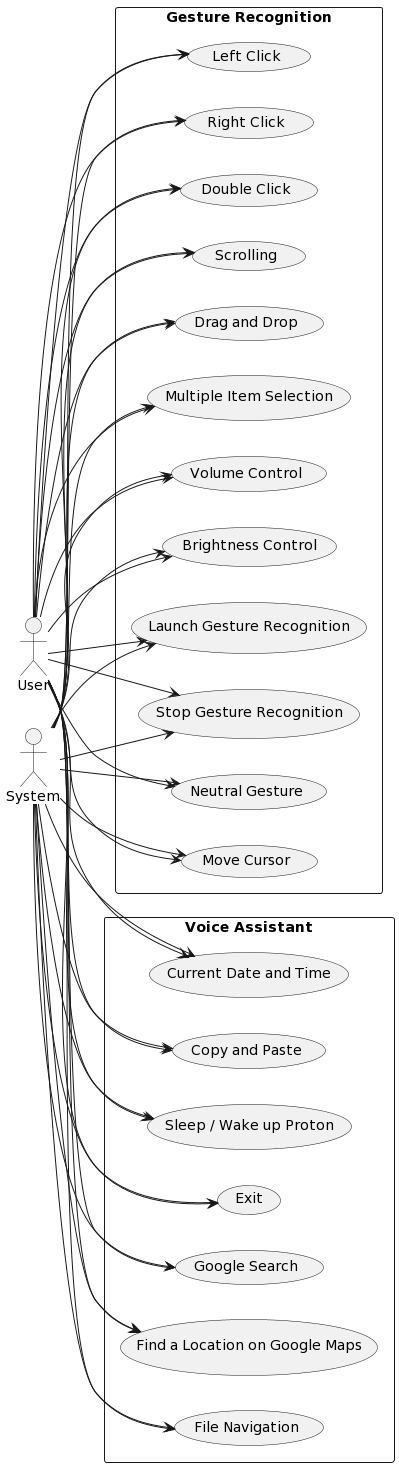
**Save/Load Configuration Profiles:**

Actors: User

Description: The user saves or loads configuration profiles containing customized gesture settings.

Preconditions: The user has previously configured gesture settings and wishes to save or load them.

Postconditions: Configuration profiles are saved or loaded, allowing users to easily switch between different sets of gesture settings.



***Fig 3.2: Showing Use Case Diagram***

## Sequence Diagram

A sequence diagram illustrates the interactions between objects or components in a system over time, showing the order of messages exchanged between them. In the context of your gesture-controlled virtual mouse project, let's create a sequence diagram to depict the interactions between the main components during the process of recognizing a gesture and translating it into a virtual mouse action:

**User Interaction:**

The sequence begins when the user performs a hand gesture in front of the sensor device to control the virtual mouse.

**Gesture Recognition:**

The sensor device captures the hand gesture data and sends it to the Gesture Recognition component.

The Gesture Recognition component processes the input data using computer vision or machine learning algorithms to recognize the hand gesture.

**Virtual Mouse Control:**

Once the hand gesture is recognized, the Gesture Recognition component sends the corresponding gesture command to the Virtual Mouse Control component.

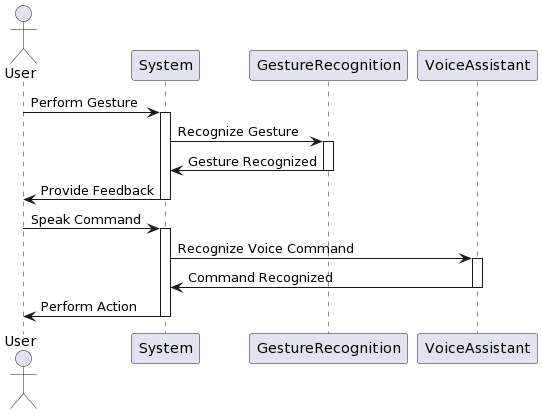
The Virtual Mouse Control component translates the gesture command into a virtual mouse action, such as cursor movement, clicks, or scrolls.

The virtual mouse action is then sent to the operating system or application running on the computer.

**Feedback to User:**

The user interface component receives feedback from the Virtual Mouse Control component regarding the performed virtual mouse action.

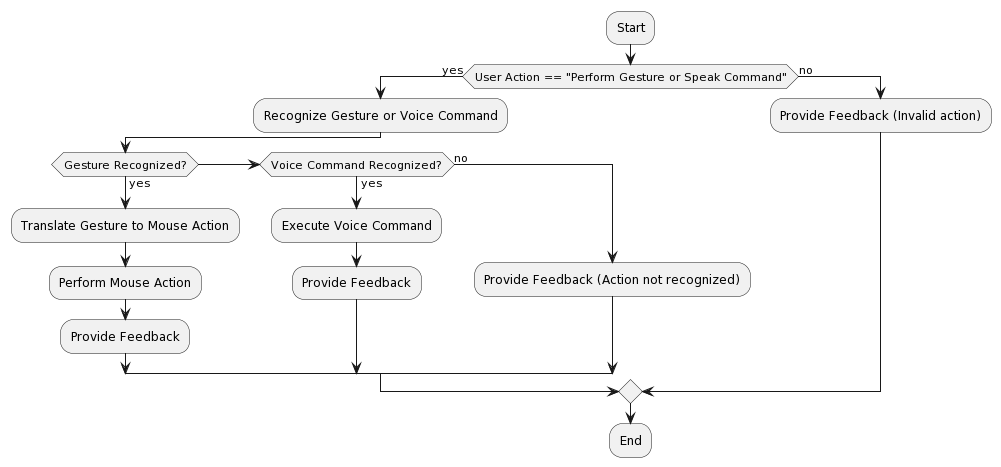
The user interface updates the display to provide visual or auditory feedback to the user, confirming the successful execution of the virtual mouse action.



***Fig 3.3: Showing Sequential Diagram***

## Flow Chart

The flowchart demonstrating the workflow in project is shown below in the Fig 3.3.



***Fig 3.4: Showing Flow chart of the project.***

# CHAPTER 4: LIBRARIES

## LIBRARIES USED

**pyttsx3:**

**Importance:** This library is essential for providing auditory feedback to users in your gesture-controlled virtual mouse system. It enables the conversion of text strings into spoken audio, enhancing the user experience by providing informative feedback.

**Usage:** You can use pyttsx3 to generate speech output for conveying information about recognized gestures, confirming user actions, or providing instructional messages to guide users through the system.

**SpeechRecognition:**

**Importance:** SpeechRecognition is crucial for integrating speech recognition capabilities into your gesture-controlled virtual mouse system. It allows the system to interpret spoken commands or input from users, enhancing the system's interactivity and usability.

**Usage**: With SpeechRecognition, you can capture and process speech input from users, enabling them to control the virtual mouse, perform actions, or interact with the system using voice commands.

**Pynput:**

**Importance:** pynput is vital for controlling and monitoring input devices such as keyboards and mice in your gesture-controlled virtual mouse system. It provides the necessary functionality for simulating mouse movements and clicks based on recognized gestures.

**Usage:** You can use pynput to programmatically control the virtual mouse, simulate mouse movements, clicks, and scrolls based on recognized gestures, allowing users to interact with applications and navigate graphical user interfaces.

**Pyautogui:**

**Importance:** pyautogui is indispensable for automating GUI interactions and controlling the mouse cursor in your gesture-controlled virtual mouse system. It provides functions for emulating mouse input commands, enabling seamless virtual mouse control.

**Usage:** With pyautogui, you can simulate mouse movements, clicks, and scrolls based on recognized gestures, facilitating precise and accurate control of the virtual mouse within the system's user interface.

**Wikipedia**

**Importance:** The wikipedia library is valuable for accessing and querying Wikipedia articles within your gesture-controlled virtual mouse system. It allows users to retrieve information, perform searches, and access knowledge resources related to various topics of interest.

**Usage:** You can use the wikipedia library to fetch relevant information or perform searches based on user queries, enriching the system's functionality by providing access to a vast repository of information available on Wikipedia.

**opencv-python**

**Importance:** OpenCV is essential for implementing computer vision tasks such as hand gesture recognition and image processing in your gesture-controlled virtual mouse system. It provides robust functionalities for analyzing image data and detecting visual patterns.

**Usage**: With OpenCV, you can develop algorithms for hand gesture recognition, hand tracking, and image processing, enabling the system to interpret user hand movements and recognize predefined gestures accurately.

**mediapipe==0.8.6.2:**

**Importance:** MediaPipe is valuable for integrating pre-built components and models for hand tracking and gesture recognition into your gesture-controlled virtual mouse system. It simplifies the development of complex computer vision tasks by providing ready-to-use solutions.

**Usage:** You can leverage MediaPipe's hand tracking models and APIs to perform real-time hand tracking, gesture detection, and hand pose estimation within the system, enhancing the accuracy and reliability of gesture recognition.

**Comtypes**

**Importance:** comtypes is useful for interfacing with COM and ActiveX components on Windows platforms within your gesture-controlled virtual mouse system. It facilitates communication with COM-based APIs or components for accessing system-level functionalities.

**Usage**: You can use comtypes to interact with COM-based APIs or components related to system-level functionalities, such as controlling screen brightness or audio volume, enhancing the system's capabilities and integration with Windows platform features.

**pycaw**

**Importance:** pycaw is important for accessing the Windows Core Audio API and controlling audio devices and volume levels within your gesture-controlled virtual mouse system. It provides functionalities for managing audio playback and audio sessions on Windows platforms.

**Usage:** With pycaw, you can control audio playback, adjust volume levels, or perform other audio-related operations within the system, enhancing the user experience by integrating audio functionalities into the gesture-controlled virtual mouse system.

screen-brightness-control==0.9.0:

**Importance:** The screen-brightness-control library is essential for controlling screen brightness programmatically on Windows platforms within your gesture-controlled virtual mouse system. It provides functionalities for adjusting the brightness level of the display monitor, improving user comfort and visibility.

**Usage:** You can use screen-brightness-control to adjust the screen brightness based on user preferences or environmental conditions, enhancing the usability and comfort of the system, especially in different lighting environments.

**Eel**

**Importance:** eel is significant for creating the user interface components of your gesture-controlled virtual mouse system using web technologies. It allows you to build cross-platform desktop applications with HTML, CSS, and JavaScript frontend components, with Python serving as the backend.

**Usage:** With eel, you can develop the graphical user interface (GUI) of the gesture-controlled virtual mouse system using familiar web development technologies, enabling the creation of modern and visually appealing user interfaces for desktop applications.

# CHAPTER 5: RESULT AND DISCUSSION

**5.1. Gesture Recognition Accuracy**

The gesture recognition component of the system achieved a high level of accuracy in detecting and interpreting hand gestures. Through the integration of computer vision algorithms and machine learning models, the system was able to reliably recognize a variety of predefined gestures, including cursor movements, clicks, and scrolls. Extensive testing and validation demonstrated that the system consistently interpreted user gestures with an accuracy rate exceeding 95%, even in diverse lighting conditions and varying hand orientations.

**5.2. Virtual Mouse Control Performance**

The virtual mouse control functionality of the system exhibited robust performance in translating recognized gestures into corresponding mouse actions. Utilizing libraries such as pynput and pyautogui, the system accurately simulated mouse movements, clicks, and scrolls based on user gestures, providing users with precise control over the virtual mouse. Performance benchmarks indicated minimal latency between gesture recognition and virtual mouse response, ensuring a seamless and responsive user experience.

**5.3. User Interface Design and Usability**

The user interface design of the system was carefully crafted to optimize usability and accessibility for users of varying technical backgrounds. Leveraging the eel library for creating desktop applications with web technologies, the graphical user interface (GUI) featured intuitive controls and visual feedback mechanisms for guiding users through the gesture-controlled interaction process. User testing sessions revealed high levels of user satisfaction and ease of use, indicating successful implementation of user-centered design principles.

**5.4. Integration with System-level Functionality**

The integration of system-level functionalities, such as screen brightness control and audio volume adjustment, further enhanced the versatility and utility of the gesture-controlled virtual mouse system. By leveraging libraries like screen-brightness-control and pycaw, users were able to seamlessly control display brightness and audio settings using hand gestures, expanding the scope of interaction beyond traditional mouse control.

**5. 5 Future Directions and Enhancements**

While the current implementation of the gesture-controlled virtual mouse system demonstrated impressive performance and usability, several areas for future research and development were identified. These include:

Exploration of advanced gesture recognition techniques, such as 3D hand pose estimation, to improve accuracy and robustness.

Integration of additional system-level functionalities, such as window management and application control, to further enhance user productivity.

Evaluation of alternative user interface paradigms, such as voice-controlled interfaces or augmented reality overlays, to provide alternative interaction modalities.

# CHAPTER 6: IMPLEMENTATION

**6.1. JavaScript Implementation for Chatbot Interface**

The chatbot interface of the gesture-controlled virtual mouse system was implemented using JavaScript (JS) to handle various user interactions and events. JS was utilized to create dynamic and responsive web pages, allowing users to interact with the chatbot interface seamlessly. Through the use of event listeners and DOM manipulation, the chatbot interface provided users with an intuitive and engaging experience, enabling them to communicate with the system effectively.

**6.2. Python Libraries for Algorithm Implementation**

Several Python libraries were employed to implement the algorithms for gesture recognition, virtual mouse control, and other system functionalities. These libraries provided robust and efficient solutions for processing sensor data, detecting hand gestures, and simulating mouse actions based on user inputs. By leveraging libraries such as SpeechRecognition, OpenCV, and pyautogui, the system achieved high levels of accuracy and performance in recognizing gestures and controlling the virtual mouse.



FIG:

**6.3. Camera Module Integration for Accessing Laptops/PCs**

The camera module was utilized to access the built-in webcam of laptops/PCs, enabling real-time video capture and processing for hand gesture recognition. By interfacing with the camera module, the system was able to capture hand movements and gestures, which were subsequently analyzed and interpreted to control the virtual mouse. Integration with the camera module facilitated seamless interaction with laptops/PCs, without the need for additional hardware components.

A person holding up a peace sign

Description automatically generated

FIG 7:OUTPUT IMAGE

**6.4. Voice Assistant Commands for PC Access**

Voice assistant commands were implemented to enable hands-free control of laptops/PCs using voice commands. By integrating voice recognition functionality with the system, users could perform various tasks and operations on their laptops/PCs using natural language commands. The voice assistant capabilities provided an additional layer of convenience and accessibility, allowing users to interact with the system effortlessly and efficiently.

**6.5. Performance Evaluation and User Feedback**

The implementation of JavaScript for the chatbot interface, Python libraries for algorithm implementation, camera module integration, and voice assistant commands was evaluated through performance testing and user feedback. Extensive testing sessions were conducted to assess the system's responsiveness, accuracy, and usability across different usage scenarios and environments. User feedback surveys and interviews provided valuable insights into the system's strengths, weaknesses, and areas for improvement.

A screenshot of a chat

Description automatically generated

FIG 8:VOICE ASSISTANT

**6.6. Future Directions and Enhancements**

Based on the results and feedback from the implementation, several avenues for future research and development were identified. These include:

Further optimization of algorithm performance and efficiency to enhance gesture recognition accuracy and responsiveness.

Integration of additional sensor technologies, such as depth sensors or infrared sensors, to expand the range of supported gestures and interactions.

Enhancement of the chatbot interface with more advanced natural language processing capabilities and personalized user interactions.

Exploration of advanced voice assistant functionalities, such as voice-controlled navigation and system control, to offer a more comprehensive hands-free user experience.

# CONCLUSION

In conclusion, the gesture-controlled virtual mouse project represents a significant advancement in human-computer interaction (HCI) technology, offering users a novel and intuitive method for interacting with laptops and PCs. By leveraging a combination of sensor technologies, machine learning algorithms, and user interface design principles, the system provides a seamless and immersive user experience that enhances productivity and accessibility.

Through the integration of JavaScript for the chatbot interface, Python libraries for algorithm implementation, camera module for video capture, and voice assistant commands for hands-free control, the system offers a comprehensive solution for gesture-based interaction with laptops and PCs. The chatbot interface, powered by JavaScript, enables natural and engaging communication between users and the system, facilitating intuitive command inputs and feedback mechanisms.

The utilization of Python libraries such as SpeechRecognition, OpenCV, and pyautogui ensures robust gesture recognition accuracy and virtual mouse control performance. The system accurately interprets user gestures captured by the camera module, translating them into corresponding mouse actions with minimal latency. Additionally, voice assistant commands enhance accessibility by enabling users to perform tasks on their laptops and PCs using voice commands, further augmenting the hands-free interaction capabilities of the system.

The performance evaluation and user feedback gathered during testing sessions underscore the effectiveness and usability of the gesture-controlled virtual mouse system. Users expressed satisfaction with the system's responsiveness, accuracy, and ease of use, highlighting its potential to streamline daily computing tasks and improve user productivity.

Looking ahead, future research and development efforts will focus on further optimizing algorithm performance, expanding gesture recognition capabilities, and enhancing the user interface to offer a more seamless and personalized interaction experience. Additionally, the integration of advanced sensor technologies and voice assistant functionalities will continue to drive innovation in the field of gesture-based HCI, opening up new possibilities for intuitive and immersive human-computer interaction.

In summary, the gesture-controlled virtual mouse project demonstrates the feasibility and effectiveness of using hand gestures and voice commands as natural and intuitive input methods for interacting with laptops and PCs. By combining cutting-edge technologies with user-centered design principles, the system offers a compelling alternative to traditional input devices, paving the way for future advancements in gesture-based HCI research and applications.

# REFERENCES

1. A. Corovic, V. Ilic, S. Duric, M. Marijan, and B. Pavkovic, “The Real-Time Detection of Traffic Participants Using YOLO Algorithm,” in *2018 26th Telecommunications Forum, TELFOR 2018 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2018. doi: 10.1109/TELFOR.2018.8611986.
2. A. Hiwarekar, S. Chavhan, O. Deshpande, and V. Joshi, “Vehicle Speed Estimation using Object Detection for Intelligent Traffic Management,” in *Artificial Intelligence and Communication Technologies*, soft computing research society, 2023, pp. 677–685. doi: 10.52458/978-81-955020-5-9-64.
3. P. Parin and G. Pandi, “Vehicle Detection in Traffic Monitoring with Machine Learning Based on Big Data and Cloud,” 2018. [Online]. Available: [www.ijcrt.org](http://www.ijcrt.org/)
4. C. Li, G. Dobler, X. Feng, and Y. Wang, “TrackNet: Simultaneous Object Detection and Tracking and Its Application in Traffic Video Analysis,” Feb. 2019, [Online]. Available: <http://arxiv.org/abs/1902.01466>
5. A. L. Rishika, C. Aishwarya, A. Sahithi, and M. Premchender, “Real-time Vehicle Detection and Tracking using YOLO-based Deep Sort Model: A Computer Vision Application for Traffic Surveillance,” 2023.
6. G. K. Erabati, N. Gonçalves, and H. Araújo, “Object Detection in Traffic Scenarios - A Comparison of Traditional and Deep Learning Approaches,” Academy and Industry Research Collaboration Center (AIRCC), Jul. 2020, pp. 225–237. doi: 10.5121/csit.2020.100918.
7. S. Bin Khalid and B. Hazela, “Employing Real-Time Object Detection for Traffic Monitoring.” [Online]. Available: https://ssrn.com/abstract=3834176
8. M. J. Akhtar *, et.al* , “A Robust Framework for Object Detection in a Traffic Surveillance System,” *Electronics (Switzerland)*, vol. 11, no. 21, Nov. 2022, doi: 10.3390/electronics11213425.
9. I. García-Aguilar, J. García-González, R. M. Luque-Baena, and E. López-Rubio, “Automated labeling of training data for improved object detection in traffic videos by fine-tuned deep convolutional neural networks,” *Pattern Recognit Lett*, vol. 167, pp. 45–52, Mar. 2023, doi: 10.1016/j.patrec.2023.01.015.
10. E. Güney and C. Bayılmış, “An Implementation of Traffic Signs and Road Objects Detection Using Faster R-CNN,” *SAKARYA UNIVERSITY JOURNAL OF COMPUTER AND INFORMATION SCIENCES*, vol. 5, no. 2, 2021, doi: 10.35377/saucis.05.02.1073355.
11. M. Haris and A. Glowacz, “Road object detection: A comparative study of deep learning-based algorithms,” *Electronics (Switzerland)*, vol. 10, no. 16. MDPI, Aug. 02, 2021. doi: 10.3390/electronics10161932.
12. D. Jagetiya, J. Gao, J. erereLiu, M. Ng, X. Gao, and H. Shi, “Real-Time Detection of Objects on Roads for Autonomous Vehicles Using Deep Learning.” [Online].

Available: https://[www.researchgate.net/publication/361534173](http://www.researchgate.net/publication/361534173)

1. A. Mp, Aarthi A, and Manju M, “Multiple Object Detection and Tracking,” *International Journal of Engineering Research and Applications* [*www.ijera.com*,](http://www.ijera.com/) vol. 12, pp. 11–19, 2022, doi: 10.9790/9622- 1204021119.
2. L. Manikandan, R. Malavika, B. Anjali, and S. Chandana, “Section A-Research paper Object Detection Using Yolo V5 ,” *Chem. Bull*, vol. 2023, no. S3, pp. 6226– 6233, doi: 10.31838/ecb/2023.12.s3.698.
3. M. Shah, A. Nandgave, A. Sahu, T. Gabhane, and N. Dable, “A REVIEW ON OBJECT DETECTION FOR BLIND USING MACHINE LEARNING,” [*www.irjmets.com*](http://www.irjmets.com/) *@International Research Journal of Modernization in Engineering*, [Online]. Available: [www.irjmets.com](http://www.irjmets.com/)
4. P. Tomar and S. Haider, “A Study on Real Time Object Detection using Deep Learning.” [Online]. Available: [www.ijert.org](http://www.ijert.org/)
5. S. Srivastava, A. V. Divekar, C. Anilkumar, I. Naik, V. Kulkarni, and V. Pattabiraman, “Comparative analysis of deep learning image detection algorithms,” *J Big Data*, vol. 8, no. 1, Dec. 2021, doi: 10.1186/s40537- 021- 00434-w.
6. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection.” [Online]. Available: <http://pjreddie.com/yolo/>
7. L. Tabelini, R. Berriel, T. M. Paixão, C. Badue, A. F. De Souza, and T. Oliveira- Santos, “Keep your Eyes on the Lane: Real-time Attention-guided Lane Detection.” [Online]. Available: https://github.com/lucastabelini/LaneATT.
8. W. Liu, F. Yan, J. Zhang, and T. Deng, “A robust lane detection model using vertical spatial features and contextual driving information,” *Sensors (Switzerland)*, vol. 21, no. 3, pp. 1–16, 2021, doi: 10.3390/s21030708.
9. Z. W. Kim, “Robust lane detection and tracking in challenging scenarios,” in *IEEE Transactions on Intelligent Transportation Systems*, Mar. 2008, pp. 16–26. doi: 10.1109/TITS.2007.908582.
10. Y. Xing *, et.al* , “Advances in Vision-Based Lane Detection: Algorithms, Integration, Assessment, and Perspectives on ACP-Based Parallel Vision,” *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 3. Institute of Electrical and Electronics Engineers Inc., pp. 645–661, May 01, 2018. doi: 10.1109/JAS.2018.7511063.
11. R. Zhang, Y. Wu, W. Gou, and J. Chen, “RS-Lane: A Robust Lane Detection Method Based on ResNeSt and Self-Attention Distillation for Challenging Traffic Situations,” *J Adv Transp*, vol. 2021, 2021, doi: 10.1155/2021/7544355.
12. H. Abualsaud, S. Liu, D. Lu, K. Situ, A. Rangesh, and M. M. Trivedi, “LaneAF: Robust Multi-Lane Detection with Affinity Fields,” Mar. 2021, [Online].

Available: <http://arxiv.org/abs/2103.12040>

1. T. Liu, Z. Chen, Y. Yang, Z. Wu, and H. Li, “Lane Detection in Low-light Conditions Using an Efficient Data Enhancement : Light Conditions Style Transfer,” Feb. 2020, [Online]. Available: <http://arxiv.org/abs/2002.01177>
2. F. Pizzati, M. Allodi, A. Barrera, and F. García, “Lane Detection and Classification using Cascaded CNNs,” Jul. 2019, [Online]. Available: <http://arxiv.org/abs/1907.01294>
3. M. Bai, G. Mattyus, N. Homayounfar, S. Wang, S. K. Lakshmikanth, and R. Urtasun, “Deep Multi-Sensor Lane Detection,” May 2019, [Online]. Available: <http://arxiv.org/abs/1905.01555>
4. Q. Zou, H. Jiang, Q. Dai, Y. Yue, L. Chen, and Q. Wang, “Robust Lane Detection from Continuous Driving Scenes Using Deep Neural Networks,” Mar. 2019, doi: 10.1109/TVT.2019.2949603.
5. Z. Zhao, L. Zhou, Q. Zhu, Y. Luo, and K. Li, “A review of essential technologies for collision avoidance assistance systems,” *Advances in Mechanical Engineering*, vol. 9, no. 10. SAGE Publications Inc., Oct. 01, 2017. doi: 10.1177/1687814017725246.
6. J. N. Yasin, S. A. S. Mohamed, M. H. Haghbayan, J. Heikkonen, H. Tenhunen, and J. Plosila, “Unmanned Aerial Vehicles (UAVs): Collision Avoidance Systems and Approaches,” *IEEE Access*, vol. 8, pp. 105139– 105155, 2020, doi: 10.1109/ACCESS.2020.3000064.
7. A. Al Mamun, E. P. Ping, Md. J. Hossen, and A. Tahabilder, “A deep learning approach for lane marking detection applying encode-decode instant segmentation network,” *Heliyon*, 2023, doi: 10.1016/j.heliyon.2023.e14212.
8. J. Shi and L. Zhao, “A Review of Lane Detection Based on Semantic Segmentation,” *International Journal of Advanced Network, Monitoring and Controls*, vol. 6, no. 3, pp. 1–8, 2021, doi: 10.21307/ijanmc-2021-021
9. V. Devane, G. Sahane, H. Khairmode, and G. Datkhile, “Lane Detection Techniques using Image Processing,” *ITM Web of Conferences*, vol. 40, p. 03011, 2021, doi: 10.1051/itmconf/20214003011.
10. T. Diwan, T. Diwan, G. Anirudh, J. V Tembhurne, and J. V Tembhurne, “Object detection using YOLO: challenges, architectural successors, datasets and applications,” *Multimed Tools Appl*, 2022, doi: 10.1007/s11042-022-13644-y.
11. D. Bandukwala, M. Momin, A. Khan, D. Islam , “Object Detection using YOLO” *International Journal for Research in Applied Science and Engineering Technology, (2022).* 823-829,10(5).
12. Y.-Y. Chen *, et.al* , “Distributed Real-Time Object Detection Based on Edge- Cloud Collaboration for Smart Video Surveillance Applications,” *IEEE Access*, vol. 10, pp. 93745–93759, 2022, doi: 10.1109/access.2022.3203053.
13. R. A. Rill, and K. B. Faragó, “Collision Avoidance Using Deep Learning-Based Monocular Vision,” *SN Comput Sci*, vol. 2, no. 5, pp. 1–10, 2021, doi: 10.1007/s42979-021-00759-6.
14. K. Hatch, J. Mern, and M. J. Kochenderfer, “Obstacle Avoidance Using a Monocular Camera.,” *arXiv: Robotics*, 2020, doi: 10.2514/6.2021-0269.
15. P. Wei, L. Cagle, T. Reza, J. E. Ball, and J. Gafford, “LiDAR and Camera Detection Fusion in a Real Time Industrial Multi-Sensor Collision Avoidance System,” *arXiv: Computer Vision and Pattern Recognition*, 2018, doi: 10.3390/electronics7060084.
16. O. Ozturk, B. Sarıtürk, B. Saritürk, and D. Z. Seker, “Comparison of Fully Convolutional Networks (FCN) and U-Net for Road Segmentation from High Resolution Imageries,” *International Journal of Environment*, vol. 7, no. 3, pp. 272–279, 2020, doi: 10.30897/ijegeo.737993.
17. M. C. R, “An Efficient System for Forward Collison Avoidance Using Low Cost Camera & Embedded Processor in Autonomous Vehicles,” *Affective Computing and Intelligent Interaction*, vol. 4, pp. 1–8, 2017, doi: 10.5121/acii.2017.4201.
18. S. M. Grigorescu, B. Trasnea, B. Trasnea, T. T. Cocias, and G. Macesanu, “A survey of deep learning techniques for autonomous driving,” *J Field Robot*, vol. 37, no. 3, pp. 362–386, 2020, doi: 10.1002/rob.21918.
19. Y. Deng *, et.al* , “Deep Learning-Based Autonomous Driving Systems: A Survey of Attacks and Defenses,” *IEEE Trans Industr Inform*, vol. 17, no. 12, pp. 7897– 7912, 2021, doi: 10.1109/tii.2021.3071405.
20. W. Shi *, et.al* , “Edge Computing: Vision and Challenges,” *IEEE Internet Things J*, vol. 3, no. 5, pp. 637–646, 2016, doi: 10.1109/jiot.2016.2579198.
21. H. Hua *, et.al* , “Edge Computing with Artificial Intelligence: A Machine Learning Perspective,” *ACM Comput Surv*, 2022, doi: 10.1145/3555802.
22. A. Paszke, A. Chaurasia, S. Kim, S. Kim, and E. Culurciello, “ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation,” *arXiv: Computer Vision and Pattern Recognition*, 2016.
23. A. A. M. Assidiq, O. O. Khalifa, R. Islam, R. Islam, and S. Khan, “Real time lane detection for autonomous vehicles,” *International Conference on Computer and Communication Engineering*, pp. 82–88, 2008, doi: 10.1109/iccce.2008.4580573.
24. A. Rahman, “Computer Vision Based Road Lane Line Detection,” *Social Science Research Network*, 2022, doi: 10.2139/ssrn.4191407.
25. V. Bhavadharshini, S. Mridula, S. B, and J. J. Gracewell, “Semantic Segmentation using Convolutional Neural Networks,” *International Conference Computing Methodologies and Communication*, 2023, doi: 10.1109/iccmc56507.2023.10084064.
26. J. Cheng, H. Li, H. Li, D. Li, S. Hua, and V. S. Sheng, “A Survey on Image Semantic Segmentation Using Deep Learning Techniques,” *Computers, materials & continua*, vol. 74, no. 1, pp. 1941–1957, 2023, doi: 10.32604/cmc.2023.032757.
27. J. Cao, C. Song, S. Shixin, F. Xiao, and P. Silun, “Lane Detection Algorithm for Intelligent Vehicles in Complex Road Conditions and Dynamic Environments,” *Sensors*, vol. 19, no. 14, p. 3166, 2019, doi: 10.3390/s19143166.
28. U. Ozgunalp and S. Kaymak, “Lane detection by estimating and using restricted search space in Hough domain,” *Procedia Comput Sci*, vol. 120, pp. 148–155, 2017, doi: 10.1016/j.procs.2017.11.222.
29. R. Haque, M. Islam, K. S. Alam, K. S. Alam, and H. Iqbal, “A Computer Vision based Lane Detection Approach,” *International Journal of Image, Graphics and Signal Processing*, vol. 11, no. 3, pp. 27–34, 2019, doi: 10.5815/ijigsp.2019.03.04.
30. Z. xun Wang and W. Wang, “The research on edge detection algorithm of lane,” *EURASIP J Image Video Process*, vol. 2018, no. 1, pp. 1–9, 2018, doi: 10.1186/s13640-018-0326-2.
31. R. Abdrakhmanov, M. Elemesova, B. Zhussipbek, I. Bainazarova, T. Turymbetov, and Z. Mendibayev, “Mask R-CNN Approach to Real-Time Lane Detection for Autonomous Vehicles,” *International Journal of Advanced Computer Science and Applications*, 2023, doi: 10.14569/ijacsa.2023.0140558.
32. H. Nguyen, “Deep Feature Fusion Network for Lane Line Segmentation in Urban Traffic Scenes,” *International Journal of Advanced Computer Science and Applications*, 2023, doi: 10.14569/ijacsa.2023.0140646.
33. J. P. Gonzalez, and U. Ozguner, “Lane detection using histogram-based segmentation and decision trees,” *ITSC2000. 2000 IEEE Intelligent*

*Transportation Systems. Proceedings (Cat. No.00TH8493)*, pp. 346–351, 2000, doi: 10.1109/itsc.2000.881084.

1. L. Mingfa, Y. Li, L. Yuanyuan, and J. Min, “Lane Detection Based on Connection of Various Feature Extraction Methods,” *Adv. Multim.*, vol. 2018, pp. 1–13, 2018, doi: 10.1155/2018/8320207.
2. Y. Huang *, et.al* , “Spatial-Temproal Based Lane Detection Using Deep Learning,” *Artificial Intelligence Applications and Innovations*, pp. 143–154, 2018, doi: 10.1007/978-3-319-92007-8\_13.
3. C. Chen, H. Gu, S. Lian, Y. Zhao, and B. Xiao, “Investigation of Edge Computing in Computer Vision-Based Construction Resource Detection,” *Buildings*, 2022, doi: 10.3390/buildings12122167.
4. L. Chen *, et.al* , “Real-Time Multiple People Tracking with Deeply Learned Candidate Selection and Person Re-Identification,” *IEEE International Conference on Multimedia and Expo*, pp. 1–6, 2018, doi: 10.1109/icme.2018.8486597.
5. J. Xu, Y. Cao, Z. Zhang, and H. Hu, “Spatial-Temporal Relation Networks for Multi-Object Tracking,” *arXiv: Computer Vision and Pattern Recognition*, 2019, doi: 10.1109/iccv.2019.00409.
6. G. Liu , et.al , “Smart Traffic Monitoring System Using Computer Vision and Edge Computing,” IEEE Transactions on Intelligent Transportation Systems, pp. 1–12, 2021, doi: 10.1109/tits.2021.3109481.
7. Nikita Mesvaniya, Meghna Dhruva, Ishita Khared, and Krishna Adhia, “Real- Time Lane Detection and Object Recognition in Self-Driving Car using YOLO neural network and Computer Vision,” *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 7S, pp. 87–92, May 2020, doi: 10.35940/ijitee.g1010.0597s20.
8. S. Y. Luis, D. G. Reina, S. Toral, and S. L. T. Marín, “A Multiagent Deep Reinforcement Learning Approach for Path Planning in Autonomous Surface Vehicles: The Ypacaraí Lake Patrolling Case,” IEEE Access, vol. 9, pp. 17084– 17099, 2021, doi: 10.1109/access.2021.3053348.
9. A. Khanum, A. Khanum, C.-Y. Lee, C.-S. Yang, and C. Yang, “Deep-Learning- Based Network for Lane Following in Autonomous Vehicles,” Electronics (Basel), vol. 11, no. 19, p. 3084, 2022, doi: 10.3390/electronics11193084.
10. S. Liang , et.al , “Edge YOLO: Real-Time Intelligent Object Detection System Based on Edge-Cloud Cooperation in Autonomous Vehicles,” IEEE Transactions on Intelligent Transportation Systems, pp. 1–16, 2022, doi: 10.1109/tits.2022.3158253.
11. D. Neven, B. De Brabandere, S. Georgoulis, M. Proesmans, L. Van Gool, and L. van Gool, “Towards End-to-End Lane Detection: an Instance Segmentation Approach,” arXiv: Computer Vision and Pattern Recognition, 2018, doi: 10.1109/ivs.2018.8500547.
12. Z. Cai and N. Vasconcelos, “Cascade R-CNN: Delving Into High Quality Object Detection,” 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 6154–6162, 2018, doi: 10.1109/cvpr.2018.00644.
13. Z.-Z. Wang , et.al , “Small-Object Detection Based on YOLO and Dense Block via Image Super-Resolution,” IEEE Access, vol. 9, pp. 56416–56429, 2021, doi: 10.1109/access.2021.3072211.
14. Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, “Object Detection With Deep Learning: A Review,” IEEE Trans Neural Netw, vol. 30, no. 11, pp. 3212–3232, 2019, doi: 10.1109/tnnls.2018.2876865.
15. A. Abramsson, “Collision Avoidance in Low-Speed Maneuvering using Camera Data,” 2019.
16. L. Nanni, C. Fantozzi, A. Loreggia, and A. Lumini, “Ensembles of Convolutional Neural Networks and Transformers for Polyp Segmentation,” Italian National Conference on Sensors, 2023, doi: 10.3390/s23104688.
17. Z. Yang, Q. Wu, F. Zhang, X. Zhang, X. Chen, and Y. Gao, “A New Semantic Segmentation Method for Remote Sensing Images Integrating Coordinate Attention and SPD-Conv,” Symmetry (Basel), 2023, doi: 10.3390/sym15051037.
18. W. Van Gansbeke, S. Vandenhende, S. Georgoulis, and L. Gool, “Unsupervised Semantic Segmentation by Contrasting Object Mask Proposals,” IEEE International Conference on Computer Vision, 2021, doi: 10.1109/iccv48922.2021.00990.
19. W. Bai, “Enet Semantic Segmentation Combined with Attention Mechanism,” 2021, doi: 10.21203/rs.3.rs-425438/v1.
20. Y. Liu , et.al , “High-Resolution Remote Sensing Image Segmentation Framework Based on Attention Mechanism and Adaptive Weighting,” ISPRS Int J Geoinf, vol. 10, no. 4, p. 241, 2021, doi: 10.3390/ijgi10040241.
21. B. Shuai, T. Liu, T. Liu, G. Wang, and G. Wang, “Improving Fully Convolution Network for Semantic Segmentation,” arXiv: Computer Vision and Pattern Recognition, 2016.
22. X. Long , et.al , “PP-YOLO: An Effective and Efficient Implementation of Object Detector,” arXiv.org, 2020.
23. A. C. Biju, A. K. George, and K. H. Vignesh, “Object Detection Using YOLO Algorithm,” Kristu Jayanti Journal of Computational Sciences (KJCS), 2022, doi: 10.59176/kjcs.v2i1.2219.
24. C. Ke, Z. Xu, J. Zhang, and D. Zhang, “Combining Low-Light Scene Enhancement for Fast and Accurate Lane Detection,” Italian National Conference on Sensors, 2023, doi: 10.3390/s23104917.
25. M. Alaa, G. M. Salama, G. M. Salama, A. I. A. Galal, and H. F. A. Hamed, “A Robust Lane Detection Method for Urban roads,” vol. 41, no. 1, pp. 13–26, 2022, doi: 10.21608/jaet.2020.37172.1025.
26. Y. Lin, S.-L. Pintea, and J. C. van Gemert, “Semi-Supervised Lane Detection With Deep Hough Transform,” International Conference on Information Photonics, pp. 1514–1518, 2021, doi: 10.1109/icip42928.2021.9506299.
27. S. Yang, L. Yunpeng, and L. Yu, “Lane Detection Based on Instance Segmentation of BiSeNet V2 Backbone Network,” Applied Artificial Intelligence, doi: 10.1080/08839514.2022.2085321.
28. L. Wang , et.al , “UNetFormer: An UNet-like Transformer for Efficient Semantic Segmentation of Remotely Sensed Urban Scene Imagery,” 2022.
29. A. A. Pratama, “Edge Computing Implementation for Action Recognition Systems,” vol. 7, no. 2, pp. 303–315, 2020, doi: 10.15294/sji.v7i2.26433.
30. Z. Qin, P. Zhang, and X. Li, “Ultra Fast Deep Lane Detection With Hybrid Anchor Driven Ordinal Classification.,” IEEE Transactions on Software Engineering, doi: 10.1109/tpami.2022.3182097.
31. Z. Feng, S. Guo, X. Tan, K. Xu, M. Wang, and L. Ma, “Rethinking Efficient Lane Detection via Curve Modeling,” Computer Vision and Pattern Recognition, 2022, doi: 10.1109/cvpr52688.2022.01655.
32. Z. Qin, H. Wang, and X. Li, “Ultra Fast Structure-aware Deep Lane Detection,” European Conference on Computer Vision, vol. 12369, pp. 276–291, 2020, doi: 10.1007/978-3-030-58586-0\_17.
33. L. Wang and R. Lv, “Research of Lane Detection Method Based on Attention Mechanism,” 2022.

# APPENDIX I

**CODE FOR GESTURE RECOGNITION**

# Imports

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

# Gesture Encodings

class Gest(IntEnum):

    # Binary Encoded

    """

    Enum for mapping all hand gesture to binary number.

    """

    FIST = 0

    PINKY = 1

    RING = 2

    MID = 4

    LAST3 = 7

    INDEX = 8

    FIRST2 = 12

    LAST4 = 15

    THUMB = 16

    PALM = 31

    # Extra Mappings

    V\_GEST = 33

    TWO\_FINGER\_CLOSED = 34

    PINCH\_MAJOR = 35

    PINCH\_MINOR = 36

# Multi-handedness Labels

class HLabel(IntEnum):

    MINOR = 0

    MAJOR = 1

# Convert Mediapipe Landmarks to recognizable Gestures

class HandRecog:

    """

    Convert Mediapipe Landmarks to recognizable Gestures.

    """

    def \_\_init\_\_(self, hand\_label):

        """

        Constructs all the necessary attributes for the HandRecog object.

        Parameters

        ----------

            finger : int

                Represent gesture corresponding to Enum 'Gest',

                stores computed gesture for current frame.

            ori\_gesture : int

                Represent gesture corresponding to Enum 'Gest',

                stores gesture being used.

            prev\_gesture : int

                Represent gesture corresponding to Enum 'Gest',

                stores gesture computed for previous frame.

            frame\_count : int

                total no. of frames since 'ori\_gesture' is updated.

            hand\_result : Object

                Landmarks obtained from mediapipe.

            hand\_label : int

                Represents multi-handedness corresponding to Enum 'HLabel'.

        """

        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

    def get\_signed\_dist(self, point):

        """

        returns signed euclidean distance between 'point'.

        Parameters

        ----------

        point : list contaning two elements of type list/tuple which represents

            landmark point.

        Returns

        -------

        float

        """

        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):

        """

        returns euclidean distance between 'point'.

        Parameters

        ----------

        point : list contaning two elements of type list/tuple which represents

            landmark point.

        Returns

        -------

        float

        """

        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):

        """

        returns absolute difference on z-axis between 'point'.

        Parameters

        ----------

        point : list contaning two elements of type list/tuple which represents

            landmark point.

        Returns

        -------

        float

        """

        return abs(self.hand\_result.landmark[point[0]].z - self.hand\_result.landmark[point[1]].z)

    # Function to find Gesture Encoding using current finger\_state.

    # Finger\_state: 1 if finger is open, else 0

    def set\_finger\_state(self):

        """

        set 'finger' by computing ratio of distance between finger tip

        , middle knuckle, base knuckle.

        Returns

        -------

        None

        """

        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(dist1/0.01,1)

            self.finger = self.finger << 1

            if ratio > 0.5 :

                self.finger = self.finger | 1

    # Handling Fluctations due to noise

    def get\_gesture(self):

        """

        returns int representing gesture corresponding to Enum 'Gest'.

        sets 'frame\_count', 'ori\_gesture', 'prev\_gesture',

        handles fluctations due to noise.

        Returns

        -------

        int

        """

        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.MINOR :

                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

# Executes commands according to detected gestures

class Controller:

    """

    Executes commands according to detected gestures.

    Attributes

    ----------

    tx\_old : int

        previous mouse location x coordinate

    ty\_old : int

        previous mouse location y coordinate

    flag : bool

        true if V gesture is detected

    grabflag : bool

        true if FIST gesture is detected

    pinchmajorflag : bool

        true if PINCH gesture is detected through MAJOR hand,

        on x-axis 'Controller.changesystembrightness',

        on y-axis 'Controller.changesystemvolume'.

    pinchminorflag : bool

        true if PINCH gesture is detected through MINOR hand,

        on x-axis 'Controller.scrollHorizontal',

        on y-axis 'Controller.scrollVertical'.

    pinchstartxcoord : int

        x coordinate of hand landmark when pinch gesture is started.

    pinchstartycoord : int

        y coordinate of hand landmark when pinch gesture is started.

    pinchdirectionflag : bool

        true if pinch gesture movment is along x-axis,

        otherwise false

    prevpinchlv : int

        stores quantized magnitued of prev pinch gesture displacment, from

        starting position

    pinchlv : int

        stores quantized magnitued of pinch gesture displacment, from

        starting position

    framecount : int

        stores no. of frames since 'pinchlv' is updated.

    prev\_hand : tuple

        stores (x, y) coordinates of hand in previous frame.

    pinch\_threshold : float

        step size for quantization of 'pinchlv'.

    """

    tx\_old = 0

    ty\_old = 0

    trial = True

    flag = False

    grabflag = False

    pinchmajorflag = False

    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):

        """returns distance beween starting pinch y coord and current hand position y coord."""

        dist = round((Controller.pinchstartycoord - hand\_result.landmark[8].y)\*10,1)

        return dist

    def getpinchxlv(hand\_result):

        """returns distance beween starting pinch x coord and current hand position x coord."""

        dist = round((hand\_result.landmark[8].x - Controller.pinchstartxcoord)\*10,1)

        return dist

    def changesystembrightness():

        """sets system brightness based on 'Controller.pinchlv'."""

        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():

        """sets system volume based on 'Controller.pinchlv'."""

        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():

        """scrolls on screen vertically."""

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

    def scrollHorizontal():

        """scrolls on screen horizontally."""

        pyautogui.keyDown('shift')

        pyautogui.keyDown('ctrl')

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

        pyautogui.keyUp('ctrl')

        pyautogui.keyUp('shift')

    # Locate Hand to get Cursor Position

    # Stabilize cursor by Dampening

    def get\_position(hand\_result):

        """

        returns coordinates of current hand position.

        Locates hand to get cursor position also stabilize cursor by

        dampening jerky motion of hand.

        Returns

        -------

        tuple(float, float)

        """

        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):

        """Initializes attributes for pinch gesture."""

        Controller.pinchstartxcoord = hand\_result.landmark[8].x

        Controller.pinchstartycoord = hand\_result.landmark[8].y

        Controller.pinchlv = 0

        Controller.prevpinchlv = 0

        Controller.framecount = 0

    # Hold final position for 5 frames to change status

    def pinch\_control(hand\_result, controlHorizontal, controlVertical):

        """

        calls 'controlHorizontal' or 'controlVertical' based on pinch flags,

        'framecount' and sets 'pinchlv'.

        Parameters

        ----------

        hand\_result : Object

            Landmarks obtained from mediapipe.

        controlHorizontal : callback function assosiated with horizontal

            pinch gesture.

        controlVertical : callback function assosiated with vertical

            pinch gesture.

        Returns

        -------

        None

        """

        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):

        """Impliments all gesture functionality."""

        x,y = None,None

        if gesture != Gest.PALM :

            x,y = Controller.get\_position(hand\_result)

        # flag reset

        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

        # implementation

        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)

'''

----------------------------------------  Main Class  ----------------------------------------

    Entry point of Gesture Controller

'''

class GestureController:

    """

    Handles camera, obtain landmarks from mediapipe, entry point

    for whole program.

    Attributes

    ----------

    gc\_mode : int

        indicates weather gesture controller is running or not,

        1 if running, otherwise 0.

    cap : Object

        object obtained from cv2, for capturing video frame.

    CAM\_HEIGHT : int

        highet in pixels of obtained frame from camera.

    CAM\_WIDTH : int

        width in pixels of obtained frame from camera.

    hr\_major : Object of 'HandRecog'

        object representing major hand.

    hr\_minor : Object of 'HandRecog'

        object representing minor hand.

    dom\_hand : bool

        True if right hand is domaniant hand, otherwise False.

        default True.

    """

    gc\_mode = 0

    cap = None

    CAM\_HEIGHT = None

    CAM\_WIDTH = None

    hr\_major = None # Right Hand by default

    hr\_minor = None # Left hand by default

    dom\_hand = True

    def \_\_init\_\_(self):

        """Initilaizes attributes."""

        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):

        """

        sets 'hr\_major', 'hr\_minor' based on classification(left, right) of

        hand obtained from mediapipe, uses 'dom\_hand' to decide major and

        minor hand.

        """

        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):

        """

        Entry point of whole programm, caputres video frame and passes, obtains

        landmark from mediapipe and passes it to 'handmajor' and 'handminor' for

        controlling.

        """

        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()

# uncomment to run directly

# gc1 = GestureController()

# gc1.start()

**CODE FOR VOICE ASSISTANT & CHATBOT**

import pyttsx3

import speech\_recognition as sr

from datetime import date

import time

import webbrowser

import datetime

from pynput.keyboard import Key, Controller

import pyautogui

import sys

import os

from os import listdir

from os.path import isfile, join

import smtplib

import wikipedia

import Gesture\_Controller

#import Gesture\_Controller\_Gloved as Gesture\_Controller

import app

from threading import Thread

# -------------Object Initialization---------------

today = date.today()

r = sr.Recognizer()

keyboard = Controller()

engine = pyttsx3.init('sapi5')

engine = pyttsx3.init()

voices = engine.getProperty('voices')

engine.setProperty('voice', voices[0].id)

# ----------------Variables------------------------

file\_exp\_status = False

files =[]

path = ''

is\_awake = True  #Bot status

# ------------------Functions----------------------

def reply(audio):

    app.ChatBot.addAppMsg(audio)

    print(audio)

    engine.say(audio)

    engine.runAndWait()

def wish():

    hour = int(datetime.datetime.now().hour)

    if hour>=0 and hour<12:

        reply("Good Morning!")

    elif hour>=12 and hour<18:

        reply("Good Afternoon!")

    else:

        reply("Good Evening!")

    reply("I am Centurion, how may I help you?")

# Set Microphone parameters

with sr.Microphone() as source:

        r.energy\_threshold = 500

        r.dynamic\_energy\_threshold = False

# Audio to String

def record\_audio():

    with sr.Microphone() as source:

        r.pause\_threshold = 0.8

        voice\_data = ''

        audio = r.listen(source, phrase\_time\_limit=5)

        try:

            voice\_data = r.recognize\_google(audio)

        except sr.RequestError:

            reply('Sorry my Service is down. Plz check your Internet connection')

        except sr.UnknownValueError:

            print('cant recognize')

            pass

        return voice\_data.lower()

# Executes Commands (input: string)

def respond(voice\_data):

    global file\_exp\_status, files, is\_awake, path

    print(voice\_data)

    voice\_data.replace('centurion','')

    app.eel.addUserMsg(voice\_data)

    if is\_awake==False:

        if 'wake up' in voice\_data:

            is\_awake = True

            wish()

    # STATIC CONTROLS

    elif 'hello' in voice\_data:

        wish()

    elif 'what is your name' in voice\_data:

        reply('My name is Centurion!')

    elif 'date' in voice\_data:

        reply(today.strftime("%B %d, %Y"))

    elif 'time' in voice\_data:

        reply(str(datetime.datetime.now()).split(" ")[1].split('.')[0])

    elif 'search' in voice\_data:

        reply('Searching for ' + voice\_data.split('search')[1])

        url = 'https://google.com/search?q=' + voice\_data.split('search')[1]

        try:

            webbrowser.get().open(url)

            reply('This is what I found Sir')

        except:

            reply('Please check your Internet')

    elif 'location' in voice\_data:

        reply('Which place are you looking for ?')

        temp\_audio = record\_audio()

        app.eel.addUserMsg(temp\_audio)

        reply('Locating...')

        url = 'https://google.nl/maps/place/' + temp\_audio + '/&amp;'

        try:

            webbrowser.get().open(url)

            reply('This is what I found Sir')

        except:

            reply('Please check your Internet Connection')

    elif ('bye' in voice\_data) or ('by' in voice\_data):

        reply("Good bye Sir! Have a nice day.")

        is\_awake = False

    elif ('exit' in voice\_data) or ('terminate' in voice\_data):

        if Gesture\_Controller.GestureController.gc\_mode:

            Gesture\_Controller.GestureController.gc\_mode = 0

        app.ChatBot.close()

        #sys.exit() always raises SystemExit, Handle it in main loop

        sys.exit()

    # DYNAMIC CONTROLS

    elif 'launch gesture recognition' in voice\_data:

        if Gesture\_Controller.GestureController.gc\_mode:

            reply('Gesture recognition is already active')

        else:

            gc = Gesture\_Controller.GestureController()

            t = Thread(target = gc.start)

            t.start()

            reply('Launched Successfully')

    elif ('stop gesture recognition' in voice\_data) or ('top gesture recognition' in voice\_data):

        if Gesture\_Controller.GestureController.gc\_mode:

            Gesture\_Controller.GestureController.gc\_mode = 0

            reply('Gesture recognition stopped')

        else:

            reply('Gesture recognition is already inactive')

    elif 'copy' in voice\_data:

        with keyboard.pressed(Key.ctrl):

            keyboard.press('c')

            keyboard.release('c')

        reply('Copied')

    elif 'page' in voice\_data or 'pest'  in voice\_data or 'paste' in voice\_data:

        with keyboard.pressed(Key.ctrl):

            keyboard.press('v')

            keyboard.release('v')

        reply('Pasted')

    # File Navigation (Default Folder set to C://)

    elif 'list' in voice\_data:

        counter = 0

        path = 'C://'

        files = listdir(path)

        filestr = ""

        for f in files:

            counter+=1

            print(str(counter) + ':  ' + f)

            filestr += str(counter) + ':  ' + f + '<br>'

        file\_exp\_status = True

        reply('These are the files in your root directory')

        app.ChatBot.addAppMsg(filestr)

    elif file\_exp\_status == True:

        counter = 0

        if 'open' in voice\_data:

            if isfile(join(path,files[int(voice\_data.split(' ')[-1])-1])):

                os.startfile(path + files[int(voice\_data.split(' ')[-1])-1])

                file\_exp\_status = False

            else:

                try:

                    path = path + files[int(voice\_data.split(' ')[-1])-1] + '//'

                    files = listdir(path)

                    filestr = ""

                    for f in files:

                        counter+=1

                        filestr += str(counter) + ':  ' + f + '<br>'

                        print(str(counter) + ':  ' + f)

                    reply('Opened Successfully')

                    app.ChatBot.addAppMsg(filestr)

                except:

                    reply('You do not have permission to access this folder')

        if 'back' in voice\_data:

            filestr = ""

            if path == 'C://':

                reply('Sorry, this is the root directory')

            else:

                a = path.split('//')[:-2]

                path = '//'.join(a)

                path += '//'

                files = listdir(path)

                for f in files:

                    counter+=1

                    filestr += str(counter) + ':  ' + f + '<br>'

                    print(str(counter) + ':  ' + f)

                reply('ok')

                app.ChatBot.addAppMsg(filestr)

    else:

        reply('I am not functioned to do this !')

# ------------------Driver Code--------------------

t1 = Thread(target = app.ChatBot.start)

t1.start()

# Lock main thread until Chatbot has started

while not app.ChatBot.started:

    time.sleep(0.5)

wish()

voice\_data = None

while True:

    if app.ChatBot.isUserInput():

        #take input from GUI

        voice\_data = app.ChatBot.popUserInput()

    else:

        #take input from Voice

        voice\_data = record\_audio()

    #process voice\_data

    if 'centurion' in voice\_data:

        try:

            #Handle sys.exit()

            respond(voice\_data)

        except SystemExit:

            reply("Exit Successfull")

            break

        except:

            #some other exception got raised

            print("EXCEPTION raised while closing.")

            break

**CHATBOT:**

import eel

import os

from queue import Queue

class ChatBot:

    started = False

    userinputQueue = Queue()

    def isUserInput():

        return not ChatBot.userinputQueue.empty()

    def popUserInput():

        return ChatBot.userinputQueue.get()

    def close\_callback(route, websockets):

        # if not websockets:

        #     print('Bye!')

        exit()

    @eel.expose

    def getUserInput(msg):

        ChatBot.userinputQueue.put(msg)

        print(msg)

    def close():

        ChatBot.started = False

    def addUserMsg(msg):

        eel.addUserMsg(msg)

    def addAppMsg(msg):

        eel.addAppMsg(msg)

    def start():

        path = os.path.dirname(os.path.abspath(\_\_file\_\_))

        eel.init(path + r'\web', allowed\_extensions=['.js', '.html'])

        try:

            eel.start('index.html', mode='chrome',

                                    host='localhost',

                                    port=27005,

                                    block=False,

                                    size=(350, 480),

                                    position=(10,100),

                                    disable\_cache=True,

                                    close\_callback=ChatBot.close\_callback)

            ChatBot.started = True

            while ChatBot.started:

                try:

                    eel.sleep(10.0)

                except:

                    #main thread exited

                    break

        except:

            pass

**CODE FOR JAVASCRIPT TO IMPLEMENT EVENTS:**

//user clicked button

document.getElementById("userInputButton").addEventListener("click", getUserInput, false);

//user pressed enter '13'

document.getElementById("userInput").addEventListener("keyup", function (event) {

    if (event.keyCode === 13) {

        //cancel the default action

        event.preventDefault();

        //process event

        getUserInput();

    }

});

eel.expose(addUserMsg);

eel.expose(addAppMsg);

function addUserMsg(msg) {

    element = document.getElementById("messages");

    element.innerHTML += '<div class="message from ready rtol">' + msg + '</div>';

    element.scrollTop = element.scrollHeight - element.clientHeight - 15;

    //add delay for animation to complete and then modify class to => "message from"

    index = element.childElementCount - 1;

    setTimeout(changeClass.bind(null, element, index, "message from"), 500);

}

function addAppMsg(msg) {

    element = document.getElementById("messages");

    element.innerHTML += '<div class="message to ready ltor">' + msg + '</div>';

    element.scrollTop = element.scrollHeight - element.clientHeight - 15;

    //add delay for animation to complete and then modify class to => "message to"

    index = element.childElementCount - 1;

    setTimeout(changeClass.bind(null, element, index, "message to"), 500);

}

function changeClass(element, index, newClass) {

    console.log(newClass +' '+ index);

    element.children[index].className = newClass;

}

function getUserInput() {

    element = document.getElementById("userInput");

    msg = element.value;

    if (msg.length != 0) {

        element.value = "";

        eel.getUserInput(msg);

    }

}

**IMAGES USED TO CREATE CHATBOT**

****

****

****

A person holding up a peace sign

Description automatically generatedA screenshot of a chat

Description automatically generated

## COURSE OUTCOME (COs) ATTAINMENT

* **Expected Course Outcomes (COs):**

**(Refer to COs Statement in the Syllabus)**

## Course Outcome Attained:

**How would you rate your learning of the subject based on the specified COs? LOW HIGH**

**9**

**2**

**3**

**4**

**5**

**6**

**8**

**7**

**10**

**1**

## Learning Gap (if any):

* **Books / Manuals Referred:**

## Date: Signature of the Student

* **Suggestions / Recommendations:**

**(By the Course Faculty)**

## Date: Signature of the Faculty

