# Control the Mouse with Hand Gesture Recognition using Computer Vision Technique

**U20ITPR02 - PROJECT WORK**

*Submitted by*

**P. SHANMUGAPRIYAN (U20IT029)**
**K. RAJ KIRAN (U20IT022)**
**M. DHIVAKAR (U20IT006)**

*Under the guidance of*

**Dr. A Kumaravel**

**Professor**

*In partial fulfillment for the award of the degree of*

**BACHELOR OF TECHNOLOGY**
**IN**
**INFORMATION TECHNOLOGY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**SCHOOL OF COMPUTING**
**BHARATH INSTITUTE OF SCIENCE AND TECHNOLOGY**
**BHARATH INSTITUTE OF HIGHER EDUCATION AND RESEARCH,**
**CHENNAI – 600073**

**MARCH - 2024**

Bharath

INSTITUTE OF HIGHER EDUCATION AND RESEARCH

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

(Vide Notification No. F.9-5/2000 - U.3. Ministry of Human Resource Development. Govt of India. dated 4th July 2002)

## DEPARTMENT OF INFORMATION TECHNOLOGY

## BONAFIDE CERTIFICATE

Certified that this Report titled **"Control the Mouse with Hand Gesture Recognition using Computer Vision Technique"** is the bonafide work of **P. SHANMUGAPRIYAN (U20IT029), K. RAJ KIRAN (U20IT022) and M. DHIVAKAR (U20IT006)** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**Head of the Department**
**Dr. K Ramesh Kumar**
Professor and Head
Department of Information Technology
Bharath Institute of Higher
Education and Research
Chennai 600073

**Project Supervisor**
**Dr. A Kumaravel**
Professor
Department of Information Technology
Bharath Institute of Higher
Education and Research
Chennai 600073

**Viva Voce Examination on ______________________**

Internal Examiner

External Examiner

# DECLARATION

We declare that this project report titled Explore the places using **“Control the Mouse with Hand Gesture Recognition using Computer Vision Technique”** submitted in partial fulfillment of the degree of **B. Tech in (Information Technology)** is a record of original work carried out by us under the supervision of **Dr. A Kumaravel**, and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

**P. SHANMUGAPRIYAN**
**(U20IT029)**

**K. RAJ KIRAN**
**(U20IT022)**

**M. DHIVAKAR**
**(U20IT006)**

**Chennai**
**/ / 2024**

## ACKNOWLEDGMENT

First, I express my sincere thankfulness to the almighty, for bestowing his blessings throughout this project work. I express my gratitude to my parents for their care, support, prayers and love.

We express our deepest gratitude to **Dr. J. Sundeep Anand** our beloved president and **Dr. E. Swetha Sundeep Anand** Managing Director for providing us the necessary facilities for the completion of my project.

I would like to express my deep gratitude to our **Dr. K. Vijaya Bhaskar Raju** Pro Chancellor**, Dr. M. Sundararajan** Vice – Chancellor (In charge), **Dr. R. Hari Prakash** Additional Registrar, **Dr. R. M. Suresh** Pro Vice – Chancellor with Additional In charge Controller of Examinations, **Dr. J. Hameed Hussain** Dean Engineering who are responsible for molding my thoughts in completing my Project.

I am thankful to the **Dr. V. Khanna** Dean-IT and Head of the Department **Dr. K. Ramesh Kumar** and all staff members of the Department of INFORMATION TECHNOLOGY, BIHER, for their encouragement and guidance throughout the course of this project work.

I wish to express my sincere profound gratitude to my esteemed and endowed Project guide **Dr. A Kumaravel** Assistant Professor for her inspiring guidance, healthy criticism, valuable suggestions, and constant encouragement throughout the period of project.

I sincerely wish to express my thanks to the staff members of Library, BIHER for their valuable during this project work. I would like to thank the authors of various journals and books whose works and results are used in the project.

**P. SHANMUGAPRIYAN**
**(U20IT029)**

**K. RAJ KIRAN**
**(U20IT022)**

**M. DHIVAKAR**
**(U20IT006)**

## ABSTRACT

Human-Computer Interaction (HCI) forms the cornerstone of our interaction with digital systems, traditionally mediated through input devices like mice and keyboards. However, recent advancements have ushered in novel methods, notably hand gestures, to bridge the gap between humans and computers. This research paper presents an innovative approach to recognize hand gestures for the purpose of controlling computers, alongside the implementation of a virtual mouse and keyboard system using Computer Vision techniques.

The primary objective of this study is to enable users to interact with computers in a more intuitive and natural manner, thereby enhancing user experience and efficiency. The proposed system leverages Computer Vision algorithms to accurately interpret hand gestures, enabling users to navigate interfaces and input text without the need for physical peripherals.

Key features of the implemented virtual mouse and keyboard include comprehensive keyboard functionality and seamless cursor movement and click events, offering users complete control over computer operations. The system's performance metrics, including recognition rate and response time for various input gestures, are meticulously evaluated, and presented in the results section.

Crucially, the efficacy of the proposed approach is validated through comparative analysis with existing state-of-the-art algorithms. Results demonstrate that the presented method outperforms alternatives, achieving an impressive accuracy rate of 95%. This underscores the potential of hand gesture recognition as a viable mechanism for revolutionizing HCI paradigms and underscores its practical feasibility in real-world applications.

Overall, this research contributes to the advancement of HCI by introducing a robust framework for hand gesture recognition and virtual interaction with computers. The findings not only showcase the system's technical capabilities but also highlight its potential to redefine the way humans engage with digital technology, paving the way for more natural and immersive computing experiences.

# LIST OF FIGURES

| FIGURE | TITLE | PAGE NO. |
|---|---|---|
| 4.2 | Hardware Requirements | 13 |
| 4.3 | Software Requirements | 13 |
| 5.3.1 | Architecture Diagram | 18 |
| 5.3.2 | Data Flow Diagram | 19 |
| 9.3 | Result | 45 |
|  | Appendix 1 (Screenshots) | 66 |

# LIST OF ABBREVIATION

| S.NO | ACRONYM | ABBREVIATION |
|---|---|---|
| 1 | HCI | Human Computer Interaction |
| 2 | PC | Personal Computer |
| 3 | VR | Virtual Reality |
| 4 | AR | Augmented Reality |
| 5 | GUI | Graphical User Interface |
| 6 | API | Application Programming Interface |
| 7 | GB | Giga Byte |
| 8 | RGB | Red, Green, Blue |
| 9 | CPU | Central Processing Unit |
| 10 | UAC | User Acceptance Testing |
| 11 | OpenCV | Opensource Computer Vision |
| 12 | AI | Artificial Intelligence |
| 13 | HGR | Hand Gesture Recognition |

# TABLE OF CONTENTS

| CHAPTER NO. | DESCRIPTION | PAGE NO. |
|---|---|---|
| | **BONAFIDE CERTIFICATE** | |
| | **DECLARATION** | |
| | **ACKNOWLEDGEMENT** | |
| | **ABSTRACT** | |
| | **LIST OF FIGURES** | |
| | **LIST OF ABBREVIATION** | |
| **1** | **INTRODUCTION** | **1** |
| 1.1 | General | 1 |
| 1.2 | Need For the Study | 2 |
| 1.3 | Objectives Of the Study | 3 |
| **2** | **LITERATURE SURVEY** | **4** |
| **3** | **SYSTEM ANALYSIS** | **7** |
| 3.1 | Existing System | 10 |
| 3.2 | Proposed System | 12 |
| **4** | **SYSTEM REQUIREMENTS** | **13** |
| 4.1 | Hardware And Software Specification | 13 |
| 4.2 | Hardware Requirements | 13 |
| 4.3 | Software Requirements | 13 |
| **5** | **SYSTEM DESIGN** | **14** |
| 5.1 | Constraints | 14 |
| 5.2 | Non-Functional Requirements | 15 |
| 5.3 | Diagram | 18 |
| **6** | **SYSTEM IMPLEMENTATION** | **20** |
| 6.1 | Introduction | 20 |
| 6.2 | Modules | 21 |
| 6.3 | Implementation Steps | 22 |
| **7** | **SOFTWARE ENVIRONMENT** | **24** |

| | | |
|---|---|---|
| **8** | **TESTING** | **29** |
| 8.1 | General | 29 |
| 8.2 | Test Data and Output | 30 |
| 8.3 | Test Strategy | 35 |
| **9** | **CONCLUSION** | **44** |
| 9.1 | General | 44 |
| 9.2 | Future Enhancement | 44 |
| 9.3 | Result | 45 |
| **10** | **SOURCE CODE** | **46** |
| | **REFERENCES** | 64 |
| | **APPENDIX 1 (SCREENSHOTS)** | 66 |

# 1. INTRODUCTION

## 1.1. GENERAL

Human-Computer Interaction (HCI) can be defined as communication between user and computer system so that both will be able to exchange information. For a long time, Keyboards and mouse are the main basis of HCI.

Human-Computer Interaction (HCI) is the interface between humans and computers. Traditionally, mouse and keyboards are used to interact with computers. An approach recently introduced to interact with computers is hand gestures. In this research paper, an approach to recognize hand gestures is introduced, and a virtual mouse and keyboard with hand gesture recognition using Computer Vision techniques are implemented.

Full keyboard features and mouse cursor movement and click events are implemented to control the computer virtually. The recognition rate and response rate of all the considered inputs are calculated and presented in the results. Human computer interaction is the interface between humans and computers with the help of fingers detection in hand gesture method based on application program interface.

This approach begins with image acquisition, utilizing webcams or depth cameras to capture images or video streams of hand gestures. These captured images undergo preprocessing stages including noise reduction, image enhancement, and background subtraction to improve their quality.

Through the utilization of webcams or depth cameras, images or video streams capturing hand movements are acquired. Preprocessing steps such as noise reduction and background subtraction are applied to enhance image quality. Feature extraction algorithms identify pertinent characteristics like hand shape and motion.

Employing machine learning or deep learning models enables gesture classification, linking recognized gestures to specific commands. This implementation represents a significant advancement in HCI, offering users an intuitive and natural means of interacting with computers, potentially revolutionizing conventional interaction paradigms.

## 1.2. NEED FOR THE STUDY

Traditional human-computer interaction (HCI) methods heavily rely on input devices like mice and keyboards. While effective, there's a growing acknowledgment that exploring alternative interaction modalities could significantly enhance HCI. Hand gesture recognition emerges as a promising avenue with the potential to revolutionize computer interfaces.

By allowing users to manipulate devices through intuitive gestures, this technology has the power to make computing more accessible, efficient, and engaging for a diverse range of users.

Introducing hand gesture recognition aims to offer users a more intuitive and efficient means of interacting with computers. Leveraging computer vision techniques, this approach interprets and responds to hand movements, enabling users to control their PCs without physical input devices.

Moreover, the adoption of hand gesture recognition has the potential to enhance computing efficiency and productivity by diminishing reliance on traditional input devices. This is especially beneficial in situations where users must multitask.

Furthermore, investigating hand gesture recognition for PC control can pave the way for advancements in related fields such as virtual reality (VR) and augmented reality (AR). These technologies heavily rely on intuitive interaction methods to create immersive experiences, and hand gesture recognition stands poised to play a pivotal role in enabling users to interact seamlessly and naturally with virtual environments.

By streamlining interactions and reducing reliance on physical input devices, hand gesture recognition has the potential to boost productivity, particularly in scenarios where users need to quickly navigate or manipulate digital content.

The adoption of hand gesture recognition opens new possibilities for innovative applications and use cases across various industries, including gaming, healthcare, education, and retail.

## 1.3. OBJECTIVES OF THE STUDY

The primary objective of this study is to investigate and implement an effective approach for hand gesture recognition within the context of Human-Computer Interaction (HCI), specifically focusing on enabling virtual mouse and keyboard functionality.

Design and implement a robust and accurate hand gesture recognition system using Computer Vision techniques. This involves developing algorithms capable of detecting and interpreting various hand gestures in real-time.

Integrate the hand gesture recognition system into a virtual mouse and keyboard interface. This interface should allow users to perform common computing tasks, such as cursor movement, clicking, and text input, using hand gestures instead of physical peripherals.

Assess the performance of the implemented system in terms of accuracy, response time, and user experience. Conduct thorough testing to measure the system's ability to accurately recognize and respond to various hand gestures and identify any areas for improvement.

Implement real-time feedback mechanisms, such as visual cues or auditory signals, to provide users with immediate feedback on the recognition and interpretation of their hand gestures, enhancing usability and interaction responsiveness.

Evaluate the system's performance in various environmental conditions, such as different lighting conditions and background complexities, to assess its robustness and adaptability.

By achieving these objectives, this study aims to contribute to the advancement of HCI by demonstrating the feasibility and efficacy of hand gesture recognition as a means of virtual interaction with computers. The findings of this research endeavour have the potential to inform the development of more intuitive and user-friendly computing interfaces, benefiting a wide range of users across different contexts and applications.

# 2. LITERATURE SURVEY

## Survey Paper - I

**Archana S. Ghotkar** and **Dr. Gajanan K. Kharate**, 'Study of Vision Based Hand Gesture Recognition Using Indian Sign Language,' *Matoshri College of Engineering and Research Centre, Department of Electronics and Telecommunication Engineering*, ISSN: 1178-5608, 2014.

- Standard sign languages (SL) are gestural languages that convey symbolic encoded messages for communication without the use of speech.
- Real-time, vision-based hand gesture recognition is increasingly feasible due to advancements in computer vision, image processing, and pattern recognition. However, its full potential for Human-Computer Interaction (HCI) has yet to be fully explored.
- A typical Hand Gesture Recognition system comprises four main modules: Gesture acquisition, Tracking and segmentation, Feature extraction and description, and Classification and recognition.
- Sign language is not universal, leading to the multidisciplinary research area of sign language recognition, which involves pattern recognition, computer vision, natural language processing, and psychology.
- HCI has become a significant focus of research, with vision-based systems gaining preference over traditional data glove-based systems due to their suitability and avoidance of cumbersome devices.
- This paper discusses a vision-based approach for interpreting the Indian sign language (ISL) using hand modality.
- Additionally, it explores the need and motivation for interpreting ISL, highlighting the potential opportunities it provides for hearing-impaired individuals in various sectors such as Industry Jobs, IT sector Jobs, and Government Jobs.

## Survey Paper - II

**Arshi Zafar** and **Prof. Asif Khan**, ‘Vision based Computer Cursor Control Mechanism by using Hand Gesture Recognition,’ *Department of CSE, Greater Noida Institute of Technology*, ISSN: 2348-2281, 2019.

- In the dynamic landscape of computer technology, significant advancements have occurred over the past decade, embedding computing into various aspects of everyday life.

- Hand gestures represent the most natural and intuitive technique for Human-Computer Interaction (HCI), offering a compelling alternative to traditional input devices such as the computer mouse.

- By harnessing vision technology and enabling mouse control through natural hand gestures, the workspace required for computing tasks can be significantly reduced, promoting efficiency and ergonomic design.

- Vision-based approaches stand out for their naturalness, as they eliminate the need for hand devices, making interaction more fluid and intuitive. Hand gestures are theoretically classified into two categories: static and dynamic gestures.

- Hand gesture recognition has been pursued through both non-vision and vision-based approaches. Non-vision-based methods include detecting finger movements with wired gloves, while vision-based techniques utilize camera systems for gesture interpretation.

- Static hand gestures maintain a consistent hand position and orientation in space over a period, while dynamic gestures involve changes in hand position or orientation within the specified timeframe.

- Examples of dynamic hand gestures include waving the hand, while static gestures encompass gestures like forming the "Ok" symbol by joining the thumb and forefinger, representing stable hand configurations.

## Survey Paper – III

**Aviral Gupta and Dr. Neeta Sharma,** ‘A Real Time Air Mouse Using Video Processing,’ *International Journal of Advanced Science and Technology*, ISSN: 2005-4238, 2020.

- Hand gesture recognition represents a simple yet innovative means of interacting with computers, offering a multi-dimensional approach to enhance user-computer interactions.
- Personal Computers (PCs) boast well-developed Graphical User Interfaces (GUIs), providing users with efficient and intuitive access to various applications via input devices like mice and trackpads.
- Touch screen technology has become ubiquitous in today's mobile phones, offering users a tactile interface for seamless interaction with their devices.
- Our approach involves leveraging video devices to create a mouse control system capable of performing all mouse tasks, including clicking (both left and right), double-clicking, and scrolling.
- While touch screens are prevalent and effective in many applications, their adoption in desktop systems is hindered by cost and hardware limitations, making them impractical alternatives.
- Our objective was to develop a virtual mouse system using a webcam, offering a user-friendly alternative to touch screens that is both cost-effective and feasible for desktop and laptop computers.
- By harnessing vision technology and enabling mouse control through natural hand gestures, we aim to reduce the workspace required for computing tasks, enhancing user comfort and efficiency.

# 3. SYSTEM ANALYSIS

Human-computer interaction (HCI) is a multidimensional process that delves into the intricate interplay between users and computer systems. It begins with a thorough examination of user requirements, encompassing factors such as user preferences, tasks, and environments.

Through this exploration, the system objectives are defined, outlining the goals and functionalities necessary to meet user needs effectively. Central to HCI is the identification of suitable input and output mechanisms. This involves selecting interfaces and interaction techniques that facilitate seamless communication between users and computers.

Factors such as accessibility, intuitiveness, and efficiency play a pivotal role in determining the appropriateness of these mechanisms. Human-computer interaction (HCI) is a multidimensional process that delves into the intricate interplay between users and computer systems. It begins with a thorough examination of user requirements, encompassing factors such as user preferences, tasks, and environments.

Through this exploration, the system objectives are defined, outlining the goals and functionalities necessary to meet user needs effectively. Central to HCI is the identification of suitable input and output mechanisms. This involves selecting interfaces and interaction techniques that facilitate seamless communication between users and computers.

Factors such as accessibility, intuitiveness, and efficiency play a pivotal role in determining the appropriateness of these mechanisms. Technical feasibility assessment is another critical aspect of HCI. This involves evaluating the capabilities of existing technologies and infrastructures to support the proposed system design.

Considerations such as hardware compatibility, software integration, and scalability are carefully weighed to ensure the viability of the HCI solution. Moreover, HCI analysis extends to the exploration of potential risks and challenges. Identifying potential obstacles such as usability issues, technical limitations, and user acceptance barriers allows for proactive mitigation strategies to be devised.

By addressing these challenges early in the design process, HCI practitioners can enhance the likelihood of project success and user satisfaction. Overall, HCI represents a holistic approach to designing interactive systems that prioritize user experience and system effectiveness.

Through a comprehensive understanding of user needs, clear definition of objectives, careful selection of interfaces, rigorous feasibility assessment, and proactive risk management, HCI endeavours to create seamless and impactful interactions between humans and computers.

**User Requirements Identification:**

Identify the primary target user group(s) for the system, considering demographic factors such as age, technical proficiency, and potential physical limitations. Consideration should be given to users who may have varying levels of familiarity with technology and those who may face accessibility challenges.

Specify the tasks and interactions users anticipate performing using hand gestures. This may include basic functionalities such as cursor control, clicking, scrolling, and text input, as well as more advanced actions like gesture-based commands for specific applications or tasks.

Utilize various methods such as surveys, interviews, or usability testing to gather feedback from potential users. Seek insights into their preferences, expectations, and any potential usability issues they may anticipate. Pay particular attention to feedback regarding gesture recognition accuracy, ease of use, and overall user experience.

Analyse the collected user feedback to identify common themes, preferences, and pain points. Use this information to refine the system requirements and prioritize features and functionalities that align with user needs and expectations.

Embrace an iterative design process that incorporates ongoing user feedback and testing.

**Functional Requirements Definition:**

Define a comprehensive range of gestures to be recognized by the system, covering common computing tasks and applications. Specify the corresponding actions or commands associated with each gesture to ensure consistency and intuitive interaction.

The system should accurately detect and interpret hand gestures in real-time. This includes recognizing a predefined set of gestures and translating them into corresponding actions or commands.

The system should enable users to control the mouse cursor and perform mouse-related actions using hand gestures. This includes functionalities such as cursor movement, clicking (both left and right), scrolling, and text input.

The system should provide users with feedback to confirm successful gesture recognition and execution. This may include visual indicators such as cursor movement or changes in interface elements, as well as auditory or haptic feedback options for users with visual impairments.

Establish performance criteria to evaluate the effectiveness of the system. This includes parameters such as gesture recognition accuracy, response time (latency between gesture input and system response), and overall system reliability (robustness in various environmental conditions and user scenarios).

Ensure compatibility with common computing platforms and applications to facilitate seamless integration into existing workflows. Consider interoperability with different operating systems, software applications, and input devices to maximize usability and accessibility for users across diverse environments.

**Technical Feasibility Assessment:**

Assess the feasibility of implementing the proposed system with available hardware resources, such as cameras and processors. Consider factors such as the resolution and frame rate capabilities of the camera, as well as the processing power and memory requirements of the system.

Evaluate the availability and suitability of software tools and libraries for implementing key functionalities, such as Computer Vision algorithms and gesture recognition techniques. Consider factors such as the compatibility of existing libraries with the chosen programming language and development environment.

Analyse the computational requirements of the proposed system, including processing power and memory usage. Ensure that the chosen hardware and software resources can meet the demands of real-time gesture recognition and system operation.

Identify potential integration challenges, such as interfacing with operating systems and application programming interfaces (APIs). Consider compatibility issues with existing hardware and software environments, as well as any constraints imposed by platform-specific requirements.

Explore strategies for optimizing resource utilization, such as efficient algorithm implementation, parallel processing techniques, and memory management strategies. Aim to minimize resource overhead while maximizing system performance and reliability.

Identify potential risks and challenges associated with the technical implementation of the proposed system. Consider factors such as hardware failures, software bugs, and compatibility issues, and develop contingency plans to mitigate these risks effectively.

Embrace an iterative development approach that allows for continuous refinement and optimization of the system design based on technical feasibility assessments and user feedback. Prioritize incremental improvements and address technical challenges proactively to ensure the successful implementation of the proposed system.

## 3.1. EXISTING SYSTEM

- Researchers have long explored the fusion of Human-Computer Interaction (HCI) principles with robotics to facilitate mouse movement control through video devices. Diverse methodologies have been employed to initiate mouse click events effectively.

- One approach involves fingertip tracking for mouse movement manipulation, wherein a user's hand passing over the screen triggers mouse clicks. Another innovative technique, as proposed by Lien, utilizes fingertip tracking to navigate the cursor.

- Certain implementations necessitate users to maintain cursor proximity to a designated area to execute a click. Notably, vision-based HCI methods exclusively rely on computer cameras for interaction, negating the requirement for supplementary hardware.

- Vision-based HCI systems, leveraging real-time object tracking and gesture recognition, offer a compelling alternative to conventional touchpad or mouse interfaces. Employing webcams, these systems track colored objects and interpret gestures for seamless interaction with the system.

- Image acquisition in existing systems utilizes webcams or depth cameras, capturing images or video streams of user hand movements to facilitate gesture recognition.

- Post image acquisition, sophisticated feature extraction algorithms are applied to identify salient characteristics of hand gestures, encompassing parameters like hand shape, motion, and spatial relationships between key points.

- The user interface of such systems often integrates calibration steps, enabling users to fine-tune gesture recognition parameters and providing guidance on executing recognized gestures accurately.

- Continuous system performance evaluation and optimization are imperative to ensure accuracy, robustness, and responsiveness in real-world scenarios. This involves iteratively collecting user feedback, updating training datasets, and refining algorithms to enhance system capabilities and user experience.

## 3.2. PROPOSED SYSTEM

- The object tracking process depicted in the system's flowchart applies continuously to each frame extracted from the real-time video stream, ensuring seamless tracking performance.

- This real-time object tracking application is built upon image processing techniques, offering a user-friendly alternative to traditional input devices like mice and keyboards by recognizing hand gestures for system control.

- Incorporating real-time feedback mechanisms, such as visual or auditory cues, provides users with immediate confirmation of gesture recognition and associated actions, enhancing overall user experience and system responsiveness.

- Adaptive learning algorithms play a crucial role in continuously refining gesture recognition models based on user feedback and usage patterns, ensuring the system's adaptability and accuracy over time.

- The exploration of integrating multiple input modalities, including voice commands or gaze tracking, alongside hand gesture recognition, enriches the interaction experience, offering users greater flexibility and versatility.

- System design prioritizes compatibility with various operating systems and software applications, facilitating seamless integration into diverse computing environments and workflows.

- Robust privacy measures are implemented to safeguard user data, ensuring secure communication channels between the gesture recognition system and the PC, bolstering user trust and confidentiality.

- Intuitive calibration tools and user interfaces empower users to personalize gesture recognition settings and gestures according to their preferences and requirements, enhancing usability and customization capabilities.

# 4. SYSTEM REQUIREMENTS

## 4.1. HARDWARE AND SOFTWARE SPECIFICATION

- Vision-Based Hand Gesture Recognition (HGR) systems are characterized by three key components: data collection, data environment, and hand gesture representation. HGR holds a prominent position in Human-Computer Interaction (HCI).

- Creating virtual human interactive systems offers a cost-effective alternative to touch screens, expanding accessibility to interactive technologies.

## 4.2. HARDWARE REQUIREMENTS

| | | |
|---|---|---|
| • Hard Disk | : | 500GB and Above |
| • RAM | : | 4GB and Above |
| • Processor | : | I3 and Above |
| • Webcam | : | One or Integrated Camera |

## 4.3. SOFTWARE REQUIREMENTS

| | | |
|---|---|---|
| • Operating System | : | Windows 10 (64 bit) |
| • Software | : | Python |
| • Tools | : | Anaconda |

# 5. SYSTEM DESIGN

## 5.1. CONTRAINTS

Constraints play a multifaceted role within various frameworks, assuming different forms and functions. They can manifest as informal textual descriptions, delineating boundaries, and guiding principles.

Alternatively, constraints may emerge as operational restrictions, imposing tangible limitations on system behavior. In some instances, constraints seamlessly integrate into pre-existing model concepts, enriching the understanding of system dynamics.

Conversely, they can also exist as standalone entities, distinctly shaping decision-making processes. Moreover, constraints often lurk implicitly within the structural fabric of models, subtly influencing outcomes and possibilities.

Whether explicit or implicit, constraints serve as crucial pillars shaping the landscape of problem-solving and decision-making paradigms.

The process begins with the determination of the classes implicated in the system, laying the foundation for subsequent analysis. Following this, a meticulous examination is conducted to identify the specific objects entwined within these classes, elucidating their roles and relationships.

Simultaneously, attention is directed towards discerning the actions intertwined within the system's operations, comprehensively mapping out the dynamic behaviors. Additionally, the identification of requisite clauses is imperative, delineating the conditions necessary for the system to function effectively.

Moreover, a holistic view encompassing global actions and constraint realization is pivotal, ensuring the seamless integration of constraints into the broader operational framework.

Through this systematic approach, a comprehensive understanding of the system's intricacies is attained, facilitating informed decision-making and robust problem-solving methodologies.

A hierarchical structuring of relations may result in more classes and a more complicated structure to implement. Therefore, it is advisable to transform the hierarchical relation structure to a simpler structure such as a classical flat one.

It is straightforward to transform the developed hierarchical model into a bipartite, flat model, consisting of classes on the one hand and flat relations on the other. Flat relations are preferred at the design level for reasons of simplicity and implementation ease.

There is no identity or functionality associated with a flat relation. A flat relation corresponds with the relation concept of entity-relationship modeling and many object-oriented methods.

- Constraints in Analysis
- Constraints in Design
- Constraints in Implementation

## 5.2. NON-FUNCTIONAL REQUIREMENTS

### 5.2.1 PERFORMANCE REQUIREMENTS

- At the heart of the proposed architecture resides an embedded browser, functioning as the cornerstone for both navigation and accessing web services. This browser serves as the primary interface through which users interact with the system, providing a seamless and intuitive platform for navigation and interaction.

- Complementing this front-end interface, the server-side infrastructure plays a pivotal role in housing the core components that drive the functionality of the architecture.

- Here, essential processes and data management operations are executed, ensuring the smooth operation and delivery of services to users. The server-side components act as the backbone of the system, handling complex computations and data manipulation tasks with efficiency and reliability.

- Furthermore, the embedded browser and server-side infrastructure work in tandem to form a cohesive framework that facilitates efficient communication, processing, and delivery of information within the system.

- The embedded browser acts as the user-facing interface, while the server-side components handle the backend operations, seamlessly integrating to provide users with a seamless and responsive experience.

- In addition to facilitating navigation and accessing web services, the embedded browser may also support advanced features such as multimedia playback, interactive forms, and real-time communication functionalities.

- These additional features enhance the user experience and expand the capabilities of the system, allowing for more dynamic and engaging interactions.

- Moreover, the server-side infrastructure may incorporate robust security measures to safeguard sensitive data and protect against potential threats. This includes encryption protocols, access control mechanisms, and intrusion detection systems to ensure the integrity and confidentiality of user information.

- Overall, the proposed architecture offers a comprehensive solution for efficient navigation, web service access, and data management, leveraging the synergies between the embedded browser and server-side infrastructure to deliver a seamless and responsive user experience.

### 5.2.2 SAFETY REQUIREMENTS

- The integrity level of software within a system holds significant importance, particularly in domains where safety is paramount. Ensuring the integrity of software becomes a top priority in such cases, as any compromise could potentially lead to safety hazards or system failures.

- Even in scenarios where software tasks may seem unrelated to safety, such as transaction logging, their role in contributing to overall system reliability cannot be overlooked. Every component of the system plays a part in upholding its integrity.

- Maintaining a high integrity level necessitates coherence across both hardware and software components. If software achieves a certain integrity standard, it becomes imperative for the supporting hardware to meet or exceed that level of integrity. This alignment ensures that the system operates reliably and safely as a whole.

- Efforts to achieve perfection in code quality are rendered futile if the underlying hardware and system software lack reliability. The integrity of the entire system is contingent upon the reliability of its individual components.

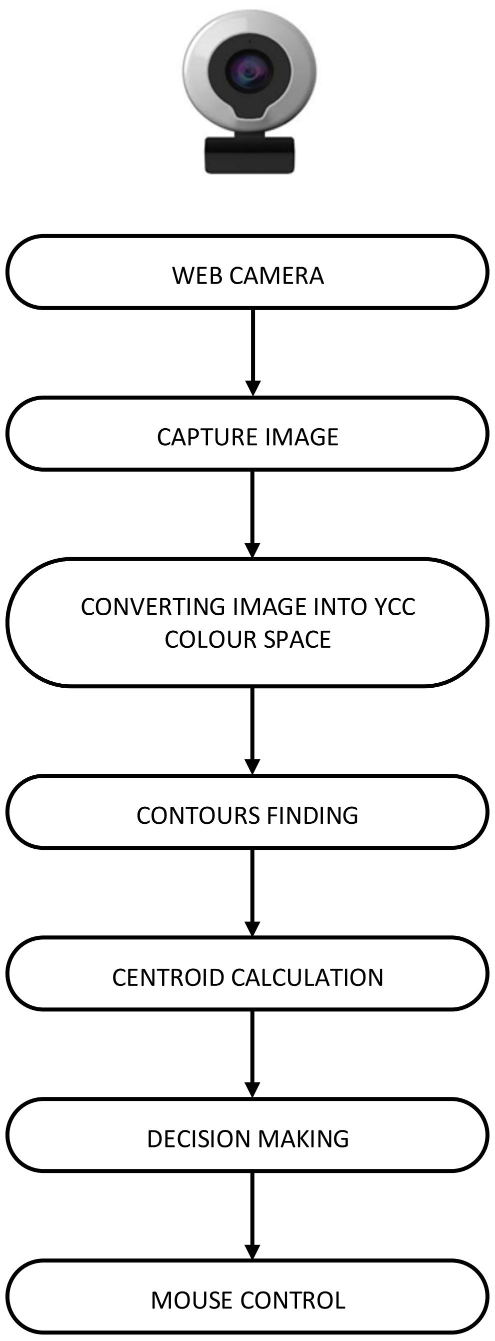
- To mitigate risks associated with lower-integrity software, it is essential to segregate high-integrity software from its lower-integrity counterparts. This segregation helps prevent the potential compromise of critical operations by software with lower integrity levels.

- The integrity of the entire system hinges not only on the integrity of its software components but also on the robustness of its hardware and system software. A holistic approach to system integrity ensures the overall reliability and safety of the system.

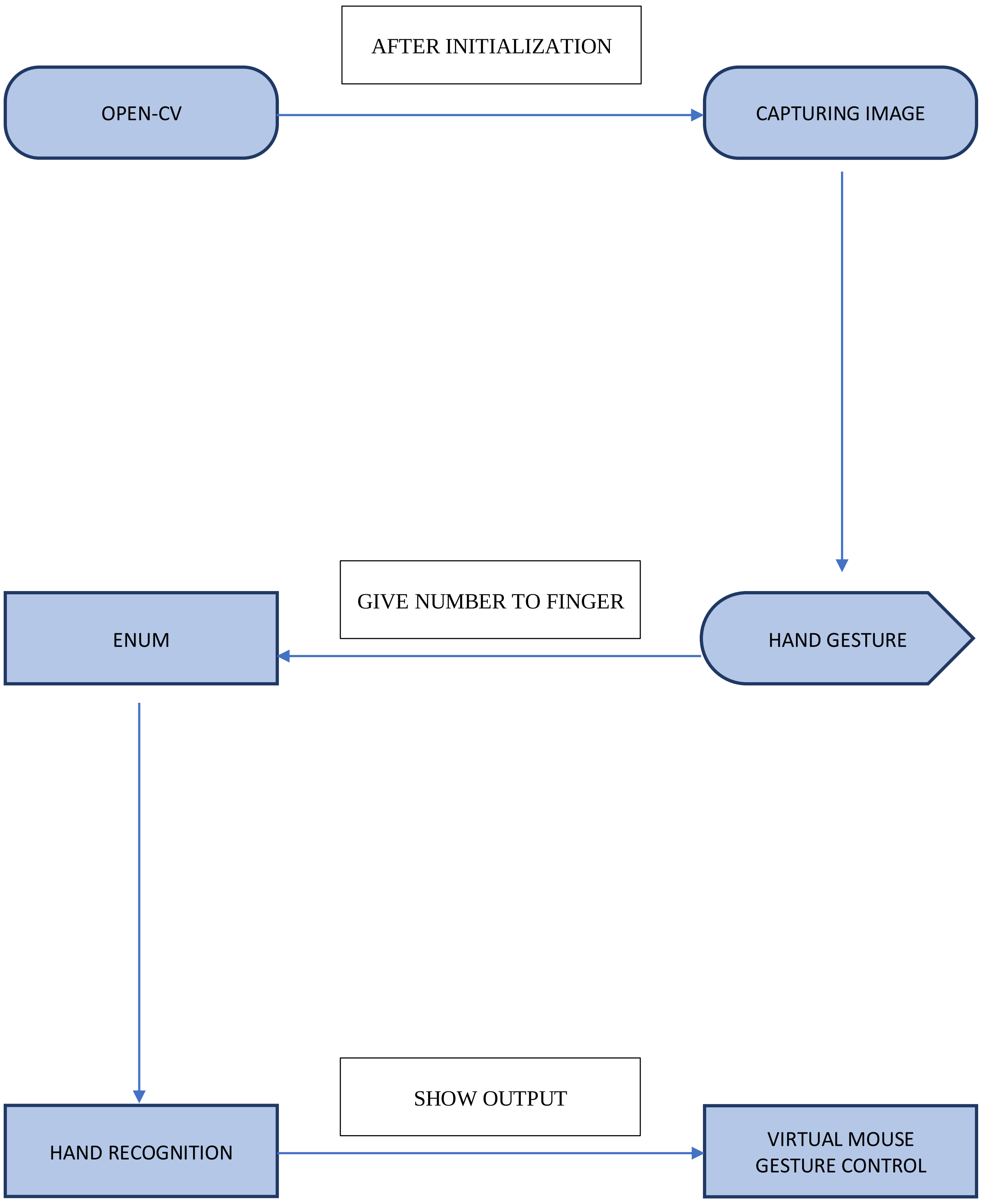
- Maintaining a high integrity level demands coherence across hardware and software.

## 5.3. DIAGRAM

### 5.3.1 ARCHITECTURE DIAGRAM



### 5.3.2 DATA FLOW DIAGRAM



# 6. SYSTEM IMPLEMENTATION

## 6.1. INTRODUCTION

The implementation of the hand gesture recognition system integrated with a virtual mouse and keyboard represents a significant advancement in the field of Human-Computer Interaction (HCI). This section provides an overview of the key components, methodologies, and technologies employed in realizing this innovative system.

- The implemented system is anchored on the development of a robust hand gesture recognition algorithm, which forms the cornerstone of its functionality. Utilizing innovative Computer Vision techniques, this component is meticulously engineered to accurately detect and interpret hand gestures in real-time, ensuring seamless interaction between users and the system.

- Expanding upon the hand gesture recognition system, the virtual mouse and keyboard interface serve as the bridge between recognized gestures and computer commands.

- By translating gestures into intuitive actions, this interface effectively replaces traditional input devices, offering users a novel and intuitive means of interacting with the system. The virtual mouse and keyboard interface replicate the functionalities of physical input devices, enabling users to perform tasks such as cursor movement, clicking, and text input using natural hand gestures.

- This intuitive interaction paradigm enhances user engagement and efficiency, facilitating smoother and more natural interaction with the computer. The seamless integration of the hand gesture recognition system with the virtual mouse and keyboard interface is facilitated by efficient data processing and communication protocols.

- These mechanisms ensure that recognized gestures are swiftly translated into system commands, minimizing latency, and maximizing responsiveness. Real-time processing capabilities are paramount in ensuring minimal latency between gesture recognition and system response.

- By leveraging real-time processing techniques, the system achieves smooth and responsive interaction, enhancing the overall user experience. User interface design plays a pivotal role in ensuring the usability and accessibility of the system.

- Intuitive interaction paradigms are employed to guide users in performing gestures and understanding system feedback, fostering a user-friendly and engaging experience.

- To validate the effectiveness and reliability of the implemented system, comprehensive performance testing and validation procedures are conducted. Through rigorous evaluation, the system's performance under various conditions is assessed.

## 6.2. MODULES

The system initiates its operation by capturing the scene through a camera, acquiring a digital representation of the environment. This initial step lays the foundation for subsequent processing and analysis.

- Upon capturing the view, the system proceeds to extract individual frames, segmenting the continuous stream of visual data into discrete snapshots.

- Within each frame, the system employs color tracking techniques to identify and track the color of the object of interest. By isolating the object from the background, the system can focus on analyzing its specific characteristics.

- Once the object is identified, the system extracts the RGB values of its constituent pixels, providing a comprehensive characterization of its color profile. This data serves as the basis for further analysis and decision-making.

- The system then compares the RGB values of the object's pixels against a predefined reference, determining whether they match or deviate.

- This comparison enables the system to discern subtle differences in color and texture, aiding in object recognition and classification.

- Based on the outcome of the comparison, the system adjusts the visual representation of the object within the frame. Pixels that match the reference are converted to white, while those that deviate are converted to black, effectively altering their visual appearance.

- By analyzing the converted pixels across the frame, the system derives insights regarding the object's presence, absence, or specific characteristics.

- These insights serve as the basis for subsequent actions or analyses, guiding the system's decision-making process.

- Following the conversion of pixels, the system may apply additional image processing techniques such as edge detection or morphology operations to further refine the object's representation and extract relevant features.

- To enhance robustness and adaptability, the system may incorporate machine learning algorithms to dynamically adjust the reference values or classification thresholds based on real-time feedback, thereby improving its accuracy and performance over time.

## 6.3. IMPLEMENTATION STEPS

- The system initiates by capturing continuous video footage from the webcam, which is subsequently transformed into a series of individual image frames. This process enables the system to analyze and process visual data frame by frame, facilitating real-time interaction.

- Each image frame undergoes a transformation from the RGB color space to grayscale, simplifying subsequent analysis and enhancing computational efficiency. By reducing the color dimensionality, the system focuses on essential visual features for object detection and tracking.

- Distinct colors within the grayscale image frames are identified and extracted, highlighting their presence, and enabling targeted analysis. This step lays the groundwork for detecting and tracking specific-colored objects within the scene.

- Contours of the colored objects are then identified within the grayscale image frames, outlining their boundaries and providing spatial information for further processing. These contours serve as a fundamental element in object localization and tracking.

- The midpoint between the centroids of the detected contours is computed, serving as a stable reference point for spatial positioning. This calculated midpoint facilitates precise tracking and provides a basis for interactive control.

- Utilizing the computed midpoint as a target location, the system tracks the movement of the mouse cursor, enabling intuitive and responsive interaction. This functionality allows users to control the cursor's position in real-time, enhancing user engagement and usability.

- In parallel, a virtual keyboard interface is generated, presenting a graphical representation of keyboard keys on the screen. This virtual interface mimics the layout of a physical keyboard, providing users with familiar input options.

- Interactions between the mouse cursor and the virtual keyboard are detected by the system, enabling keyboard input without the need for physical keys. Users can simulate key presses by clicking on the virtual keys, facilitating seamless text input and system navigation.

# 7. SOFTWARE ENVIRONMENT

## 7.1. CODING

Once the design aspect of the system is finalized, the system progresses into the coding and testing phase. During the coding phase, the conceptualized design is translated into functional code in the chosen programming language(s). This stage is crucial as it brings the system to life, implementing the envisioned functionalities and features.

A paramount consideration during the coding phase is maintaining a good coding style. This entails adhering to established coding conventions, such as consistent naming conventions, proper indentation, and clear commenting. By following a standardized coding style, the codebase becomes more readable, maintainable, and scalable.

Furthermore, adopting a modular approach to coding facilitates flexibility and ease of maintenance. Breaking down the system into smaller, reusable modules promotes code reusability and facilitates easier debugging and troubleshooting.

Continuous testing and validation are integral parts of the coding phase. Unit tests, integration tests, and system tests are conducted to ensure that the code functions as expected and meets the specified requirements. Any discrepancies or errors identified during testing are addressed promptly through debugging and code optimization.

Moreover, version control systems, such as Git, are utilized to manage code changes and track revisions effectively. Version control enables collaborative development, allowing multiple developers.

## 7.2. CODING STANDARDS

Coding standards serve as guidelines for programming, focusing on the physical structure and appearance of the code. By adhering to these standards, developers can ensure that the codebase is consistent, readable, and maintainable. During the coding phase, these standards are applied to translate the blueprint developed during the design phase into executable code.

The primary purpose of coding standards is to make the code easier to read, understand, and maintain. Consistent naming conventions, indentation styles, and commenting practices contribute to code clarity and comprehensibility. This consistency enables developers to quickly grasp the functionality of the code and navigate through it efficiently.

Moreover, coding standards promote code reusability and modularity by encouraging the use of structured programming techniques and design patterns. By breaking down complex functionalities into smaller, reusable components, developers can create code that is easier to test, debug, and maintain over time.

A key aspect of coding standards is ensuring that the code is well-documented. Clear and concise comments should be included throughout the codebase to provide insights into the logic, purpose, and functionality of each section of code. This documentation not only aids in understanding the code but also facilitates knowledge transfer among team members.

Additionally, coding standards should be designed in such a way that any programmer, regardless of their familiarity with the codebase, can understand the code and make changes as needed. This ensures that the codebase remains flexible and adaptable to evolving requirements and future enhancements.

Overall, adhering to coding standards is essential during the coding phase as it promotes code quality, readability, and maintainability. By following established coding conventions and best practices, developers can create robust, scalable, and efficient software systems. Some of the standard needed to achieve the above-mentioned objectives are as follows:

- Simplifying program design by prioritizing simplicity, clarity, and ease of understanding aids in efficient comprehension and maintenance of the codebase.

- Employing consistent and descriptive naming conventions for variables, functions, and other elements enhances code readability and facilitates comprehension by developers.

- Adhering to standard conventions for assigning and manipulating values promotes consistency and predictability in program behavior, reducing the likelihood of errors.

- Documenting scripts thoroughly and strategically utilizing comments throughout the codebase assists in understanding and maintaining the code, especially for future modifications or troubleshooting.

- Structuring message boxes according to standardized formats, incorporating relevant information and clear messaging, improves user experience and simplifies the troubleshooting process.

- Implementing robust exception and error handling mechanisms ensures the program can gracefully handle unexpected situations, enhancing its reliability and resilience in real-world scenarios.

### 7.2.1. NAMING CONVENTIONS

Naming conventions play a crucial role in ensuring the clarity and comprehensibility of code. It is imperative that classes, data members, member functions, procedures, and other elements are named in a self-descriptive manner, allowing developers to understand their purpose and scope briefly. These conventions are not only adopted to enhance readability but also to facilitate effective communication of the intended message to users, making it customary to adhere to them.

- Class names should reflect problem domain equivalence and adhere to a camel-case convention, beginning with a capital letter and utilizing mixed cases for subsequent words. This approach helps to distinguish classes and conveys their significance within the system architecture.

- The names of member functions and data members should begin with a lowercase letter, with subsequent words capitalized, and the rest of the letters in lowercase. This convention aids in differentiating these elements from class names and promotes consistency throughout the codebase. By following this convention, developers can easily discern the purpose and functionality of each member function and data member, enhancing code readability and maintainability.

### 7.2.2. VALUE CONVENTIONS

Value conventions are essential for maintaining consistency and reliability in the values assigned to variables throughout the program execution. This ensures that variables hold meaningful and valid data at any given point in time. Achieving this involves several key practices:

- Assigning appropriate default values to variables ensures that they are initialized to a sensible state before being explicitly set by the program. Default values serve as fallbacks in cases where no explicit value is provided, preventing unexpected behavior or errors.

- Validating the values entered fields, such as user inputs or data retrieved from external sources, is crucial for ensuring data integrity and preventing erroneous or malicious inputs. Proper validation mechanisms, such as range checks, format validation, and input sanitization, help maintain the integrity and reliability of the data stored in variables.

- Flag values, which often represent Boolean states or control flags, should be documented clearly to convey their intended meaning and usage. Documentation should describe the significance of each flag value, its implications on program behavior, and any dependencies or interactions with other variables or components.

### 7.2.3. SCRIPT WRITING AND COMMENTING STANDARD

Script writing is akin to an art form, where every line of code is carefully crafted to convey a specific purpose. Indentation serves as a vital tool in this creative process, as it helps organize the structure of the script and improves readability.

By properly aligning conditional and looping statements, developers can enhance the clarity of the code, making it easier to understand the flow of execution. Moreover, comments play a crucial role in guiding readers through the script, providing insights into the rationale behind certain decisions and clarifying complex sections of code.

These comments serve as signposts, helping developers navigate through the script and understand its intricacies. They also serve as valuable aids during code review and maintenance, minimizing surprises and facilitating collaboration among team members. Script writing involves a balance between functionality and comprehensibility.

By paying meticulous attention to indentation, commenting, and overall code structure, developers can create scripts that are not only functional but also accessible and easy to maintain.

### 7.2.4. MESSAGE BOX FORMAT

When communicating information to users, clarity is paramount to ensure they can understand the message effectively. To achieve this goal, a specific format has been adopted for displaying messages.

This may include organizing information into distinct sections, using clear and concise language, and providing relevant context to aid understanding.

Additionally, standardized formatting reduces ambiguity and confusion, allowing users to focus on the content of the message rather than deciphering its presentation. They are as follows:

- XUser has performed illegal operation.
- ! Information to the user.

# 8. TESTING

## 8.1. GENERAL

Testing is an essential process aimed at identifying errors and ensuring quality assurance throughout the development and maintenance lifecycle. It serves as a critical component in the validation and verification of software systems, playing a pivotal role in ensuring that the specified requirements are accurately incorporated into the design, and that the design itself is correct.

During the testing phase, the primary goal is to verify the accuracy and completeness of the design, as well as to detect any logic faults or discrepancies before proceeding with coding. Detecting and addressing design faults early in the process is crucial, as the cost of rectifying such faults increases significantly once coding has commenced. Various methods, such as inspection and walkthroughs, are employed to identify design faults and ensure the integrity of the software design.

Testing encompasses a range of activities and techniques, each aimed at uncovering diverse types of errors and evaluating the behavior of the software system. Some of the key test cases involved in the testing process include:

- Static analysis techniques are utilized to examine the structural properties of the source code without executing it. This involves analyzing the code for syntax errors, potential vulnerabilities, coding standards compliance, and other structural aspects to ensure code quality and reliability.

- Dynamic testing involves executing the program on test data to observe its behavior and assess its functionality. This technique evaluates the runtime behavior of the software, identifying bugs, logic errors, performance issues, and other defects that may arise during execution.

## 8.2. TEST DATA AND OUTPUT

### 8.2.1. UNIT TESTING

Unit testing is a critical aspect of the software development process, aimed at verifying the functional performance of individual modular components within the software. This testing approach focuses on evaluating the behavior and functionality of the smallest units of the software design, typically at the level of individual functions, methods, or classes.

During unit testing, each modular component is tested in isolation, allowing developers to identify and address defects or discrepancies within the specific unit. This isolation enables thorough testing of the unit's functionality and ensures that it behaves as expected, regardless of its integration with other components.

White box testing techniques are commonly employed for unit testing, wherein the internal structure and implementation details of the software component are considered during testing.

### 8.2.2. FUNCTIONAL TESTS

Functional testing is a crucial aspect of software testing, focusing on verifying the functionality of the software system by exercising its various features and capabilities. Functional test cases are designed to validate the behavior of the software under different scenarios, including nominal input values, boundary values, and special conditions. These test cases aim to ensure that the software behaves as expected and meets the specified requirements. Three types of tests in Functional test:

- Performance testing evaluates the responsiveness, scalability, and reliability of the software under various workload conditions. This type of test assesses the system's ability to handle a specific level of workload and measure its response time, throughput, and resource utilization.

- Stress testing involves subjecting the software system to extreme conditions beyond its normal operational capacity. This test evaluates the system's robustness and resilience by simulating high loads, excessive user traffic, or resource constraints. Stress tests aim to identify potential failure points, assess system stability under adverse conditions, and determine the system's ability to recover gracefully from failures.

- Structure testing, also known as structural testing, focuses on verifying the internal structure and logic of the software code. This type of test examines the code's implementation details, such as control flow, data flow, and program logic, to ensure correctness and completeness. Structure tests may include techniques such as code coverage analysis, path testing, and branch coverage testing to assess the adequacy of test coverage and identify code defects.

### 8.2.3. PERFORMANCE TEST

Performance testing measures various aspects of a software system's performance, including execution time, program throughput, response time, and device utilization. These metrics provide insights into the efficiency, scalability, and reliability of the system under different conditions. Here is how each aspect is determined:

- Performance testing assesses the time taken by various parts of the unit or the entire program to execute specific operations or tasks. By measuring the execution time, testers can identify bottlenecks, optimize code, and improve overall system performance.

- Throughput refers to the rate at which the software system processes requests or transactions within a given time. Performance testing evaluates the system's ability to handle concurrent users or transactions and measures the throughput achieved under various load conditions.

- Response time measures the time taken for the system to respond to a user request or input. Performance testing assesses response time across different scenarios, including normal, peak, and stress conditions, to ensure that the system meets specified performance requirements and provides a satisfactory user experience.

- Device utilization evaluates the extent to which system resources, such as CPU, memory, disk I/O, and network bandwidth, are utilized during program execution. Performance testing monitors resource consumption and identifies potential resource constraints or contention issues that may impact system performance.

### 8.2.4. STRESS TEST

Stress testing is a critical aspect of software testing aimed at evaluating the robustness and resilience of a program under extreme conditions. Unlike other types of testing that focus on verifying expected behavior, stress testing intentionally subjects the program to conditions that exceed its normal operational capacity.

The primary goal of stress testing is to identify weaknesses, vulnerabilities, and failure points in the software system by pushing it to its limits. During stress testing, the program is subjected to elevated levels of load, concurrency, or resource constraints to simulate real-world scenarios.

This testing approach helps assess the system's ability to handle peak loads, sudden spikes in user traffic, or resource shortages without compromising performance, stability, or data integrity. By intentionally pushing the program beyond its breaking point, testers can uncover potential defects, memory leaks, performance bottlenecks, or other issues that may lead to system crashes, data corruption, or degraded performance.

Analyzing how the program breaks under stress provides valuable insights into its strength, limitations, and areas for improvement. Stress testing enables developers and testers to proactively address potential failure points, optimize system performance, and enhance overall reliability and robustness.

By identifying and mitigating vulnerabilities before deployment, stress testing helps ensure that the software system can withstand the rigors of real-world usage.

### 8.2.5. STRUCTURED TEST

Structure testing, also known as white-box testing, is focused on examining the internal logic and execution paths of a program to ensure thorough test coverage. White box testing strategies, such as path testing and branch coverage, are employed to guarantee that all independent paths within a module are exercised at least once. Here are some techniques commonly used in structure testing to achieve comprehensive test coverage:

- Exercise Logical Decisions are designed to exercise all logical decisions within the code, ensuring that both true and false conditions are evaluated. This verifies the correctness of conditional statements and ensures that all outcomes are considered.

- Loop Testing is created to execute loops at their boundaries and within their operational bounds. This helps verify the correctness of loop constructs, including loop initialization, iteration, and termination conditions. By testing loop boundaries, potential off-by-one errors, and boundary conditions are identified.

- Data Structure Testing is developed to exercise internal data structures used within the program, ensuring their validity and integrity. This involves testing data structure operations such as insertion, deletion, and retrieval to validate their functionality and behavior.

- Attribute Checking cases verify the correctness of attributes and properties associated with program elements. This includes checking variable values, object attributes, and other program attributes to ensure they meet specified requirements and constraints.

- Error Handling cases are designed to verify the program's behavior in error conditions, including end-of-file conditions, I/O errors, buffer overflows, and textual errors in output information. Error handling mechanisms are tested to ensure that the program responds appropriately to exceptional situations and maintains robustness and reliability.

By employing these structure testing techniques, testers can systematically verify the internal logic and behavior of the program, identifying defects, vulnerabilities, and areas for improvement.

This approach ensures comprehensive test coverage and helps enhance the quality, reliability, and maintainability of the software system.

**8.2.6. INTEGRATION TESTING**

Integration testing is a pivotal phase in software development aimed at constructing and validating the program's structure while uncovering errors related to interfacing between modules.

It encompasses the comprehensive testing of the interconnected set of modules that constitute the software product, ensuring seamless interaction and functionality across components. The primary objective of integration testing is to identify critical modules and ensure their robustness as early as possible in the development lifecycle.

Traditionally, integration testing approaches have evolved from unstructured testing methodologies applied to small-scale programs. One approach involves combining all units once they have individually passed testing, followed by comprehensive testing of the integrated system.

Alternatively, a more incremental approach entails constructing the product in increments of tested units. This incremental integration strategy involves integrating a small set of modules, testing their interaction, and gradually adding additional modules while testing in combination.

One of the familiar challenges encountered during integration testing is linking errors, where modules fail to properly connect with supporting files, resulting in runtime issues. To address this, rigorous testing and validation of module interconnections are conducted, with errors localized and corrected within the newly integrated modules.

By systematically identifying and resolving interfacing issues early in the development process, integration testing plays a crucial role in mitigating risks and delivering a robust, high-quality software solution.

By conducting integration testing in incremental stages, developers can detect and address interface discrepancies and integration errors early in the development cycle. This iterative approach promotes early feedback and continuous improvement, leading to more efficient problem resolution and a smoother overall development process.

Integration testing serves as a crucial checkpoint for verifying the compatibility and interoperability of modules within the software system.

## 8.3. TEST STRATEGY

### 8.3.1. TESTING

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet –undiscovered error. A successful test is one that uncovers an as-yet- undiscovered error.

System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the complete set of programs hangs together.

System testing requires a test that consists of several key activities and steps for running program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created, and the documentation and related data structures are designed. Software testing is essential for correcting errors.

Otherwise, the program or the project is not said to be complete. Software testing is the critical element of software quality assurance and represents the ultimate review of specification design and coding.

Testing is the process of executing the program with the intent of finding the error. A good test case design is one that has a probability of finding a yet undiscovered error. A successful test is one that uncovers a yet undiscovered error.

#### 8.3.1.1. WHITE BOX TESTING

This testing is also called Glass box testing. In this testing, by knowing the specific functions that a product has been designed to perform a test can be conducted that demonstrates each function is fully operational at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

- Flow graph notation is used to represent the control flow of the program, depicting the sequence of execution paths and decision points within the code.

- Cyclomatic complexity is a quantitative measure of the complexity of a program's control flow. It is calculated based on the number of independent paths through the code and serves to determine the testing effort required.

- Test cases are derived from the control flow graph, aiming to cover all paths and decision points within the code. This ensures comprehensive test coverage and helps identify potential errors or vulnerabilities.

- Graph matrices, such as adjacency matrices or incidence matrices, are used to represent relationships between nodes and edges in the control flow graph. These matrices facilitate the systematic derivation of test cases and aid in identifying redundant or unreachable code paths.

### 8.3.1.2. BLACK BOX TESTING

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh,” that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

- Graph-based testing techniques utilize graphical representations, such as control flow graphs or data flow graphs, to analyze the structure and behavior of the software.

- Equivalence partitioning is a technique used to divide the input domain of a software component into equivalence classes, where each class represents a set of inputs that should produce the same output.

- Comparison testing involves comparing the output of the software under test with the expected output or with the output of a reference implementation. This technique is particularly useful for validating complex computations or algorithms where the expected output can be determined analytically or obtained from a trusted source.

Overall, glass box testing with basis path testing provides insights into the internal structure and behavior of the software, allowing testers to uncover hidden defects.

### 8.3.2. SOFTWARE TESTING STRATEGIES:

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned and conducted systematically. For this reason, a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

- Testing is a multifaceted process that unfolds incrementally, starting at the module level and progressively extending to encompass the integration of the entire computer-based system.

- This systematic approach involves the application of various testing techniques at various stages of the development lifecycle to ensure comprehensive coverage and thorough evaluation of the software's functionality and performance.

- As testing progresses, the scope expands to encompass the integration of modules, where interactions between different components are scrutinized to assess their compatibility and interoperability.

- Different testing techniques are employed at each stage of the testing process, tailored to the specific objectives and requirements of the testing phase. For instance, unit testing is commonly used during module-level testing to verify the functionality of individual units of code, while integration testing focuses on evaluating the interactions and interfaces between integrated components.

- Similarly, system testing assesses the behavior of the entire system, ensuring that it meets the specified requirements and functions as intended in its operational environment.

- Both the developer of the software and an independent test group play integral roles in the testing process. While developers are responsible for conducting unit testing and verifying the correctness of their code, independent test groups provide impartial evaluation and validation of the software from an external perspective.

#### 8.3.2.1. INTEGRATION TESTING:

Integration testing serves as a systematic approach to constructing the program structure while simultaneously uncovering errors associated with the integration of individual modules.

While modules may function correctly in isolation, combining them introduces complexities and potential interface errors that must be addressed through thorough testing.

One of the key challenges in integration testing is ensuring that individual modules, which are highly prone to interface errors, seamlessly interact when integrated into the larger system.

The process of "putting them together" requires careful consideration of how data flows between modules and how their functions interact with one another. Interface errors can manifest in various forms, including data loss, inconsistent behavior, and unexpected outcomes when modules are combined.

Interfacing modules may reveal discrepancies that were not apparent during individual module testing. For example, assumptions about data format or communication protocols may lead to compatibility issues when modules are integrated.

Similarly, variations in input data or execution paths between modules can amplify minor discrepancies to unacceptable levels, highlighting the importance of comprehensive integration testing.

Testing these interactions helps identify potential conflicts and ensures the integrity of shared data throughout the system.

**8.3.2.2. PROGRAM TESTING:**

Program testing serves as a crucial phase in software development, aimed at identifying both logical and syntax errors that may compromise the functionality and reliability of the program.

Syntax errors are violations of the rules defined by the programming language and are typically detected by the compiler or interpreter during the compilation or execution process.

Common examples of syntax errors include improperly defined field dimensions or missing keywords, which trigger error messages generated by the computer.

In contrast, logic errors pertain to incorrect data processing or decision-making within the program. These errors may manifest as out-of-range values, invalid combinations of data, or incorrect calculations.

Unlike syntax errors, logic errors may not be detected by the compiler and require manual examination of the program's output to identify discrepancies.

Condition testing is a method used to exercise the logical conditions present in a program's code. These conditions may include Boolean operators, variables, parentheses, relational operators, or arithmetic expressions.

By rigorously testing conditions and evaluating their outcomes, developers can identify and rectify both syntax and logic errors, thereby improving the overall quality and reliability of the software.

Condition testing serves as a valuable technique for ensuring the correctness and robustness of the program's logic, contributing to the successful deployment of error-free software systems.

#### 8.3.2.3. SECURITY TESTING:

Security testing is a critical aspect of software development aimed at verifying the effectiveness of security measures implemented within a system. Its primary goal is to assess whether the system's protection mechanisms can safeguard it against unauthorized access, data breaches, and other security threats.

One key aspect of security testing is evaluating the system's resilience to both frontal and rear attacks. Frontal attacks typically involve direct attempts to breach the system's defenses through known vulnerabilities or weaknesses.

Rear attacks, on the other hand, focus on exploiting less obvious entry points or vulnerabilities that may not be immediately apparent. During security testing, the tester assumes the role of a potential attacker and employs various techniques and tools to simulate real-world threats.

This may include conducting penetration testing, vulnerability assessments, and ethical hacking activities to identify and exploit vulnerabilities in the system's security posture.

By adopting the perspective of an attacker, security testers can effectively assess the system's susceptibility to diverse types of attacks and vulnerabilities. This proactive approach helps uncover potential security weaknesses and allows developers to address them before they can be exploited by malicious actors.

Overall, security testing plays a crucial role in ensuring the confidentiality, integrity, and availability of the system, helping to mitigate security risks and protect sensitive information from unauthorized access or disclosure.

#### 8.3.2.4. VALIDATION TESTING

At the conclusion of integration testing, the software is fully assembled into a cohesive package, ensuring that all individual modules function harmoniously together.

During this process, any interfacing errors or compatibility issues between modules are identified and rectified. Subsequently, the software undergoes a final round of validation testing to ensure that it meets the expectations and requirements of the customer.

Software validation verifies that the software behaves in a manner consistent with the customer's expectations and specifications. It confirms that the software fulfills its intended purpose and delivers the desired functionality.

This validation process is typically conducted through a series of black-box tests, which assess the software's behavior without delving into its internal.

- If the software behaves as expected and meets the specified requirements during the validation testing, it is considered to have successfully passed validation. This indicates that the software is ready for deployment and use by the end-users.

- Conversely, if the software fails to meet the specified requirements or behaves unexpectedly during validation testing, it is deemed to have failed validation. In such cases, further investigation and corrective actions are necessary to address the discrepancies and ensure that the software aligns with the customer's expectations.

At this stage of the project, any deviations or errors identified during validation testing are addressed promptly to ensure the successful completion of the project. Collaborative efforts between the development team and the user community are crucial in resolving these deficiencies effectively.

Through negotiation and discussion, a method for addressing and rectifying the identified issues is established, allowing for timely resolution.

Despite encountering deficiencies in the system during validation testing, it is important to note that these issues were not deemed catastrophic. While they may have impacted certain aspects of the system's functionality or performance, they were not severe enough to jeopardize the overall success of the project. Instead, they served as valuable learning opportunities, highlighting areas for improvement and refinement.

By actively engaging with users and stakeholders to address these deficiencies, the development team demonstrates a commitment to delivering a high-quality and robust solution that meets the needs and expectations of the end-users.

Through collaboration and continuous improvement, the proposed system undergoes iterative refinement, leading to its successful deployment and adoption.

### 8.3.2.5. USER ACCEPTANCE TESTING

User acceptance testing (UAT) is a critical phase in the development process, ensuring that the system meets the needs and expectations of its intended users.

Throughout the development lifecycle, close collaboration with prospective users is maintained to solicit feedback and incorporate changes as needed. Key aspects of user acceptance testing include input screen design and output screen design.

- During UAT, particular attention is given to the design of input screens to ensure they are intuitive, user-friendly, and aligned with the workflow of end-users. Prospective users are actively involved in reviewing and providing feedback on input screen layouts, data entry fields, dropdown menus, and other interactive elements.

- Similarly, the design of output screens is evaluated during user acceptance testing to ensure that the information presented meets the needs of users and is displayed in a clear and comprehensible manner. Users assess the layout, formatting, and content of output screens to determine if they provide the necessary insights and support decision-making processes effectively.

- By actively involving users in the testing and validation of input and output screen designs, the development team gains valuable insights into user preferences, pain points, and usability issues. This collaborative approach fosters user buy-in and confidence in the system, contributing to its successful adoption and utilization.

# 9. CONCLUSION

## 9.1. GENERAL

In conclusion, the implementation of a novel object tracking-based keyboard and mouse system represents a significant advancement in human-computer interaction. Developed using Python and OpenCV, this system offers a cost-effective solution for controlling computers and gaming consoles using hand gestures captured by a webcam.

The system's versatility extends to augmented reality applications, gaming, and even third person gaming experiences, enhancing immersion and accessibility. One of the key advantages of this technology is its potential to assist individuals with impaired muscle control, providing them with a viable means of interacting with digital devices without the need for expensive additional hardware.

With the ongoing collaboration between developers, researchers, and end-users, we can expect to see continued progress in enhancing the ways in which humans interact with computers and digital environments.

## 9.2. FUTURE ENHANCEMENT

- In the future, the integration of artificial intelligence (AI) holds immense promise for advancing hand gesture recognition (HGR) systems. By leveraging AI-powered trained classifiers, it becomes possible to detect and interpret hand gestures with greater accuracy and efficiency.

- However, this approach presents certain challenges that must be addressed. One major obstacle is the requirement for vast amounts of data to train the classifier effectively. Gathering and annotating such datasets can be resource-intensive and time-consuming.

- Another challenge lies in selecting the distinguishing characteristics of the object being recognized, which requires careful consideration and experimentation to achieve optimal results.

## 9.3. RESULT

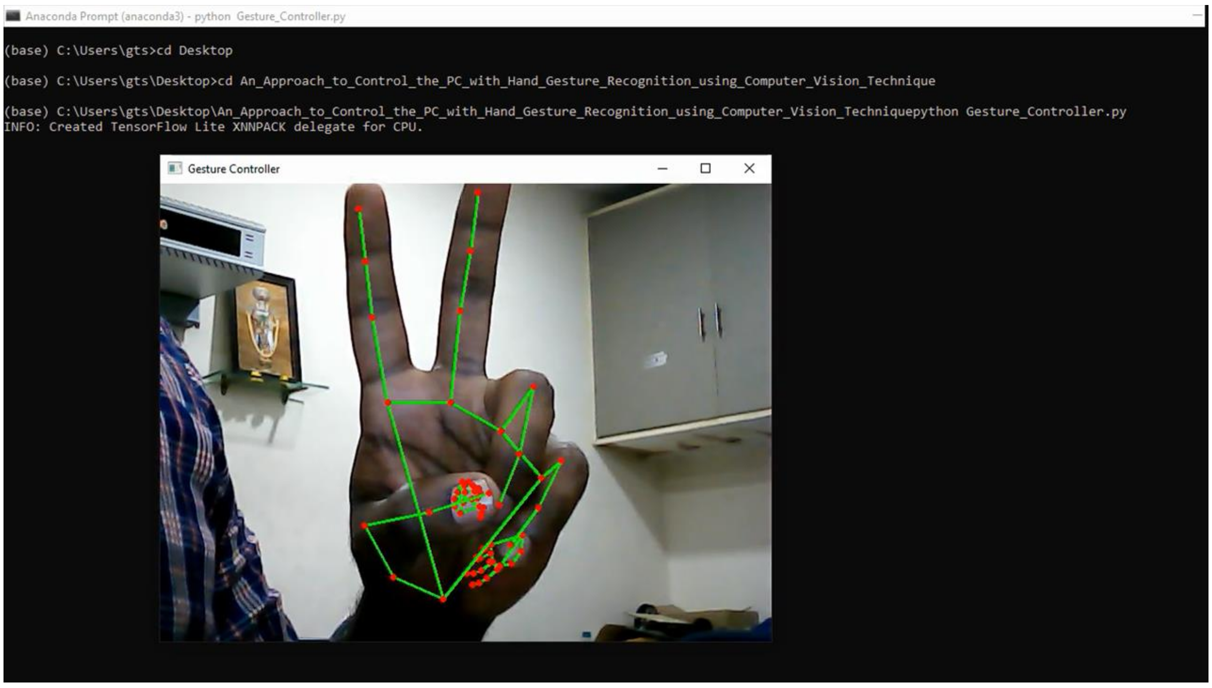
The implementation of the object tracking-based keyboard and mouse system using the PC's webcam, coupled with the utilization of Python and OpenCV, has yielded promising results. Through rigorous testing and evaluation, several key outcomes have been observed:

- The developed system demonstrates robust performance in accurately detecting and interpreting hand gestures in real-time.

- Various hand gestures, including cursor movement, clicking, and text input, were successfully recognized with high precision.

- The integration of hand gesture recognition into a virtual mouse and keyboard interface proved to be effective, offering users intuitive control over computer interactions.

In summary, the results of this project underscore the effectiveness and feasibility of utilizing hand gesture recognition for human-computer interaction.



```
(base) C:\Users\gts>cd Desktop

(base) C:\Users\gts\Desktop>cd An_Approach_to_Control_the_PC_with_Hand_Gesture_Recognition_using_Computer_Vision_Technique

(base) C:\Users\gts\Desktop\An_Approach_to_Control_the_PC_with_Hand_Gesture_Recognition_using_Computer_Vision_Techniquepython Gesture_Controller.py
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
```

# 10. SOURCE CODE

```
import cv2
import mediapipe as mp
import pyautogui
import math
from enum import IntEnum
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from google.protobuf.json_format import MessageToDict
import screen_brightness_control as sbcontrol

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

class Gest(IntEnum):
    """
    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
    V_GEST = 33
    TWO_FINGER_CLOSED = 34
    PINCH_MAJOR = 35
    PINCH_MINOR = 36

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

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

        formula = d = 2√(a2+b2)
        """
        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)
```

```
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

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

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')

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

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()

        elif Controller.pinchdirectionflag == False:
            controlVertical()

    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)

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

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

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

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

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

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

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

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

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

'''
---------------------------------------- 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
    hr_minor = None
    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(1) & 0xFF == ord('x')):
                    break
        GestureController.cap.release()
        cv2.destroyAllWindows()

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

## REFERENCES

- **Mohamed, N., Mustafa, M. B., & Jomhari, N. (2021).** A Review of the Hand Gesture Recognition System: *Current Progress and Future Directions. IEEE Access.*

- **Guo, L., Lu, Z., & Yao, L. (2021).** Human-machine interaction sensing technology based on hand gesture recognition: *A review. IEEE Transactions on Human-Machine Systems.*

- **Sharma, N., Mangla, M., Mohanty, S. N., & Satpathy, S. (2021).** A Gesture based Remote Control for Home Appliances. *In 2021 8th International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 42-47). IEEE.*

- **Agarwal, D., Rastogi, A., Rustagi, P., & Nijhawan, V. (2021).** Real Time RF-Based Gesture Controlled Robotic Vehicle. *In 2021 8th International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 848-852). IEEE.*

- **Pendke, K., Khuje, P., Narnaware, S., Thool, S., & Nimje, S. (2015).** Computer cursor control mechanism by using hand gesture recognition. *International Journal of Computer Science and Mobile Computing, 4(3), 293-300.*

- **Erdem, A., Erdem, E., Yardimci, Y., Atalay, V., & Cetin, A. E. (2002).** Computer vision-based mouse. *In 2002 IEEE International Conference on Acoustics, Speech, and Signal Processing (Vol. 4, pp. IV-4178). IEEE.*

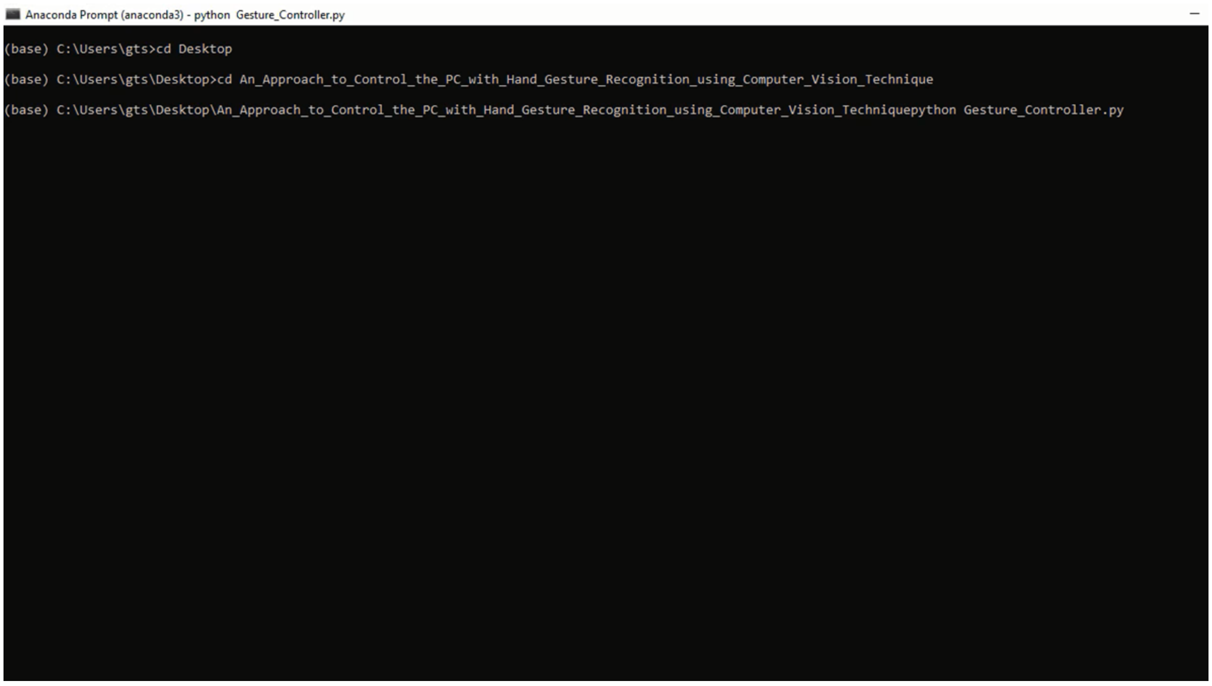
- **Park, H. (2008)**. A method for controlling the mouse movement using a real time camera. *Brown University, Providence, RI, USA, Department of computer science.*

- **Banerjee, A., Ghosh, A., Bharadwaj, K., & Saikia, H. (2014).** Mouse control using a web camera based on colour detection. *arXiv preprint arXiv:1403.4722.*

- **Lee, C. C., Shih, C. Y., & Jeng, B. S. (2010)**. Fingertip- Writing Alphanumeric Character Recognition for Vision-Based Human Computer Interaction. *In Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference* *on (pp. 533-537). IEEE.*

- **Igorevich, R. R., Park, P., Min, D., Park, Y., Choi, J., & Choi, E. (2010).** Hand gesture recognition algorithm based on grayscale histogram of the image. *In Application of Information and Communication Technologies (AICT), 2010 4th International Conference on (pp. 1-4). IEEE.*

# APPENDIX 1 (SCREENSHOTS)

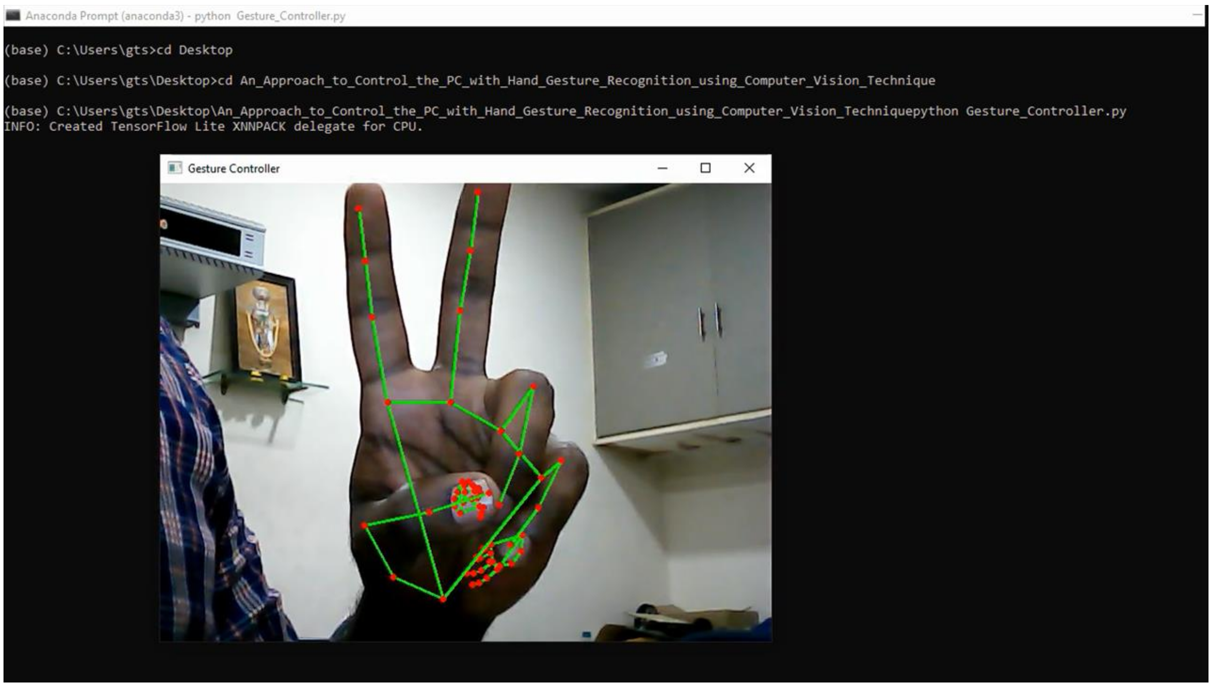


Anaconda Prompt (anaconda3) - python Gesture_Controller.py

```
(base) C:\Users\gts>cd Desktop

(base) C:\Users\gts\Desktop>cd An_Approach_to_Control_the_PC_with_Hand_Gesture_Recognition_using_Computer_Vision_Technique

(base) C:\Users\gts\Desktop\An_Approach_to_Control_the_PC_with_Hand_Gesture_Recognition_using_Computer_Vision_Techniquepython Gesture_Controller.py
```

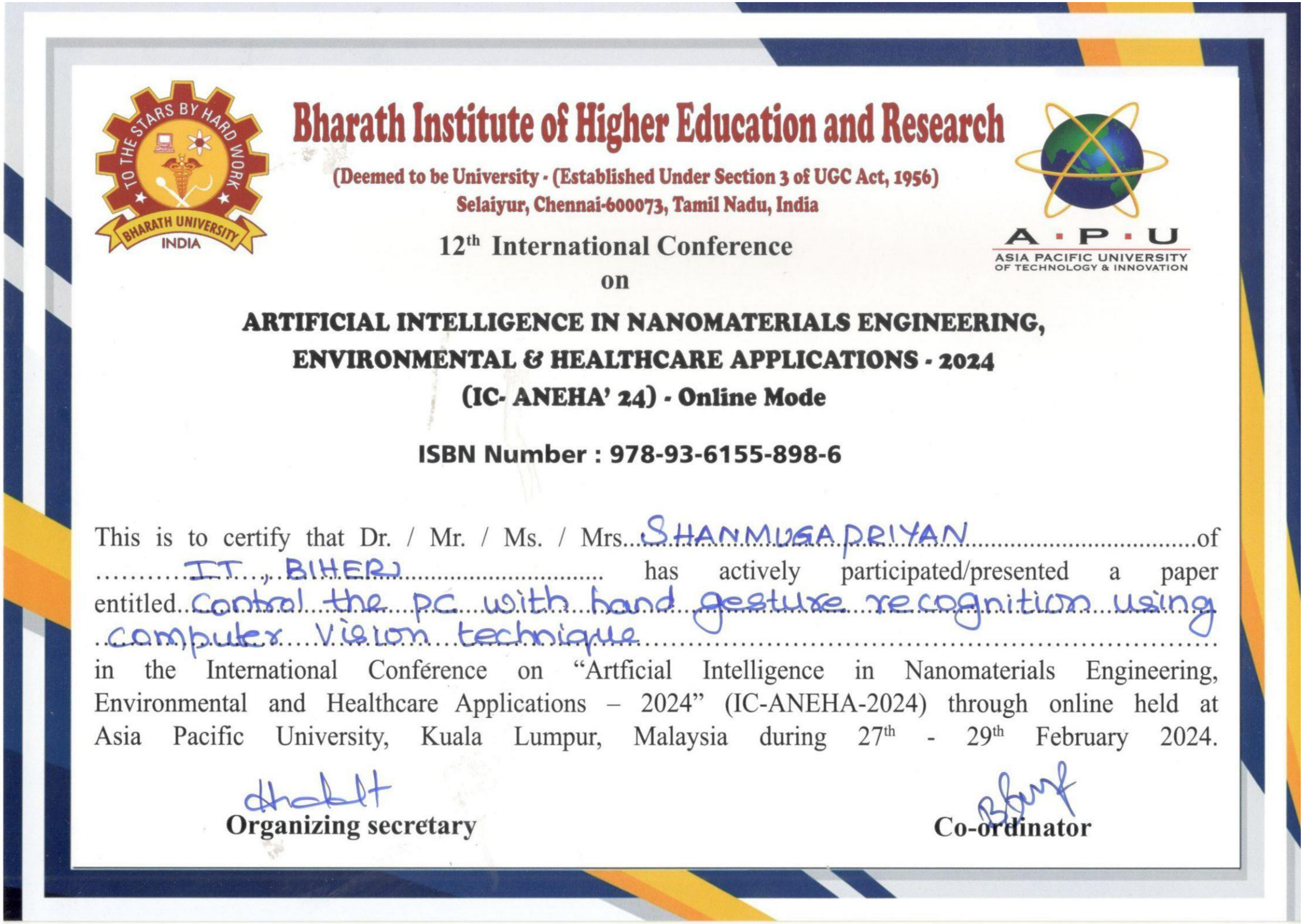


Anaconda Prompt (anaconda3) - python Gesture_Controller.py

```
(base) C:\Users\gts>cd Desktop

(base) C:\Users\gts\Desktop>cd An_Approach_to_Control_the_PC_with_Hand_Gesture_Recognition_using_Computer_Vision_Technique

(base) C:\Users\gts\Desktop\An_Approach_to_Control_the_PC_with_Hand_Gesture_Recognition_using_Computer_Vision_Techniquepython Gesture_Controller.py
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
```



TO THE STARS BY HARD WORK
BHARATH UNIVERSITY
INDIA

# Bharath Institute of Higher Education and Research

(Deemed to be University - (Established Under Section 3 of UGC Act, 1956)
Selaiyur, Chennai-600073, Tamil Nadu, India

A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

12th International Conference
on

**ARTIFICIAL INTELLIGENCE IN NANOMATERIALS ENGINEERING, ENVIRONMENTAL & HEALTHCARE APPLICATIONS - 2024**

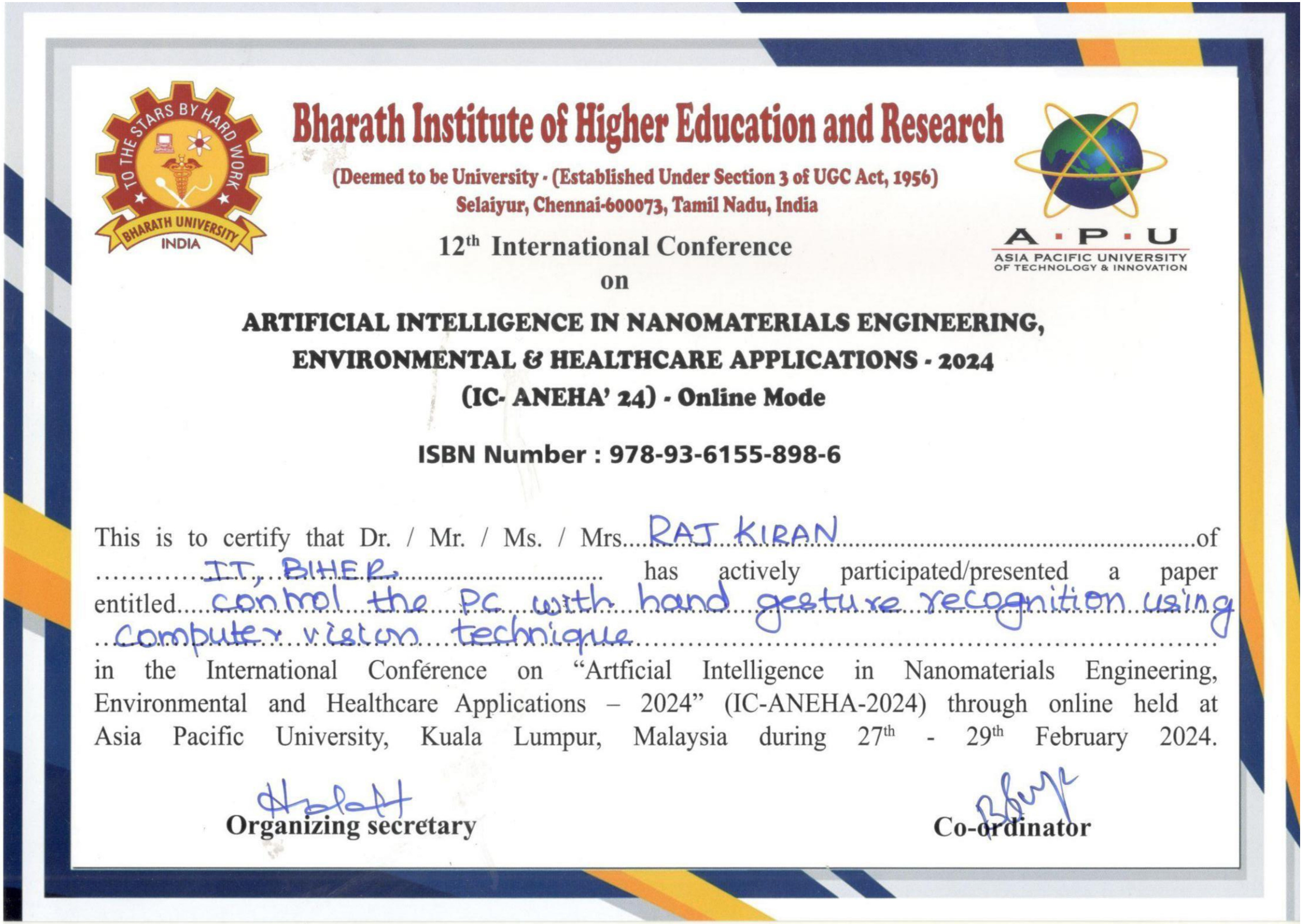
**(IC- ANEHA' 24) - Online Mode**

**ISBN Number : 978-93-6155-898-6**

This is to certify that Dr. / Mr. / Ms. / Mrs. SHANMUGAPRIYAN of IT, BIHER has actively participated/presented a paper entitled control the pc with hand gesture recognition using computer vision technique in the International Conférence on "Artficial Intelligence in Nanomaterials Engineering, Environmental and Healthcare Applications – 2024" (IC-ANEHA-2024) through online held at Asia Pacific University, Kuala Lumpur, Malaysia during 27th - 29th February 2024.

[illegible]
**Organizing secretary**

[illegible]
**Co-ordinator**



# Bharath Institute of Higher Education and Research

(Deemed to be University - (Established Under Section 3 of UGC Act, 1956)
Selaiyur, Chennai-600073, Tamil Nadu, India

A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

12th International Conference
on

**ARTIFICIAL INTELLIGENCE IN NANOMATERIALS ENGINEERING, ENVIRONMENTAL & HEALTHCARE APPLICATIONS - 2024**
**(IC- ANEHA' 24) - Online Mode**

**ISBN Number : 978-93-6155-898-6**

This is to certify that Dr. / Mr. / Ms. / Mrs. RAJ KIRAN of IT, BIHER. has actively participated/presented a paper entitled control the PC with hand gesture recognition using computer vision technique in the International Conférence on "Artficial Intelligence in Nanomaterials Engineering, Environmental and Healthcare Applications – 2024" (IC-ANEHA-2024) through online held at Asia Pacific University, Kuala Lumpur, Malaysia during 27th - 29th February 2024.

[illegible]
**Organizing secretary**

[illegible]
**Co-ordinator**

# Bharath Institute of Higher Education and Research

(Deemed to be University - (Established Under Section 3 of UGC Act, 1956)
Selaiyur, Chennai-600073, Tamil Nadu, India

A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

12th International Conference
on

**ARTIFICIAL INTELLIGENCE IN NANOMATERIALS ENGINEERING, ENVIRONMENTAL & HEALTHCARE APPLICATIONS - 2024**
**(IC- ANEHA' 24) - Online Mode**

**ISBN Number : 978-93-6155-898-6**

This is to certify that Dr. / Mr. / Ms. / Mrs. DHIVAKAR of IT, BIHER has actively participated/presented a paper entitled control the pc with hand gesture recognition using computer vision technique in the International Conférence on "Artficial Intelligence in Nanomaterials Engineering, Environmental and Healthcare Applications – 2024" (IC-ANEHA-2024) through online held at Asia Pacific University, Kuala Lumpur, Malaysia during 27th - 29th February 2024.

[illegible]
**Organizing secretary**

[illegible]
**Co-ordinator**