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

School of Computer Science Engineering and Information Systems
M.Tech (Integrated) Software Engineering
FALL 2024-2025
Project Report

AI VIRTUAL MOUSE

SWE 1901 : Technical Answers for Real World Problems (TARP)

Offered during FALL 2024-2025

(Dr. B.Prabadevi)

by

Joel Joshua Richard	**21MIS0116**
Pranay Gorantla	**21MIS0123**
Yashwanth R	**21MIS0285**
Dev Sandeep G.V	**21MIS0316**
Praveen Chidambaram P	**21MIS0343**

NOVEMBER 2024

School of Computer Science Engineering and Information Systems
M.Tech (Integrated) Software Engineering
FALL 2024-2025
< Project Title >

TEAM Number : 13
Team Member(s) with Reg # and Name : Joel Joshua Richard ; 21MIS0116 ; 9849067258 ; joeljoshua.richard2021@vitstudent.ac.in Pranay Gorantla ; 21MIS0123 ; 8309436368 ; pranay.gorantla2021@vitstudent.ac.in Yashwanth R ; 21MIS0285 ; 8438772502 ; yashwanth.2021@vitstudent.ac.in Dev Sandeep G.V ; 21MIS0316; 9597390316 ; devsandep.gv2021@vitstudent.ac.in Praveen Chidambaram P ; 21MIS0343 ; 9597532428 ; praveenchidambaram.p2021@vitstudent.ac.in
Project Title : AI VIRTUAL MOUSE

1. Introduction

1.1 Background (System Study Details in brief)

The AI Virtual Mouse using finger tracking aims to revolutionize human-computer interaction by providing a more intuitive and immersive interface. Traditional input devices like the mouse and keyboard are limited in dynamic and immersive environments such as gaming, virtual reality (VR), and augmented reality (AR). By leveraging finger tracking technology, this project offers a touchless, hands-free solution for controlling digital environments, making interactions more seamless and engaging.
As VR and AR technologies grow, the need for intuitive control mechanisms becomes crucial. Finger tracking enables precise, natural control, enhancing user experience and eliminating the need for physical peripherals. This technology also improves accessibility, particularly for individuals with physical disabilities, by allowing them to interact with digital environments without relying on traditional input devices.
The project is motivated by the increasing trend toward AR and MR applications, where finger tracking can serve as an essential interface tool. It provides a more efficient, responsive, and personalized interaction model, paving the way for more immersive and user-friendly digital experiences. By addressing the limitations of conventional input devices, this project aims to contribute to the future of touchless and adaptive technology.

1.2 Problem Statement

To create a touchless AI-based virtual mouse using finger tracking technology, enabling seamless control of digital environments. This aims to replace traditional input devices, enhancing user experience, especially in gaming, virtual reality, and augmented reality. The system should offer precise, natural, and intuitive interactions without physical peripherals.

1.3 Abstract

This project develops an AI-based virtual mouse system that uses finger tracking technology to enable touchless human-computer interaction. Through advanced computer vision and machine learning algorithms, users can control digital interfaces with natural hand gestures, eliminating the need for

traditional input devices like a mouse or keyboard. The system is designed to offer high accuracy, low latency, and seamless integration across various applications, including gaming, virtual reality (VR), and augmented reality (AR). By enhancing accessibility and user experience, it provides a more intuitive and immersive interaction model. Ultimately, this project aims to pave the way for future touchless interfaces in evolving digital environments.

1.4 SDG goal Alignment Justification

SDG 9: Industry, Innovation, and Infrastructure.

With SDG 9, AI virtual mouse and keyboard project is aligned because enhancing human-computer interaction,the area critical for technological innovation, will be encouraged as touch-free ways of interaction with digital devices are encouraged. Accessible and adaptive interfaces which could contribute to improvement in productivity and accessibility. This will align to SDG 9 in making smarter, user-centered digital infrastructure, demonstrating how intelligent systems can be used both in service of the individual and of industry. Doing this makes your work drive the advancement of the future for interactive technologies, making digital systems more inclusive and adaptive to a diverse society.

2. Related Works

2.1 Literature Survey (Should be elaborately discussed with its citation)

Hand gesture recognition has seen significant advancements in recent years, with diverse methods and technologies being explored to improve both accuracy and usability. A comprehensive review of the state of hand gesture recognition systems highlights the rapid progress made in computer vision, sensor-based techniques, and deep learning approaches, while pointing out the need for more robust machine learning models to tackle challenges such as dynamic environments and signal variability. The review also emphasizes the lack of standardized datasets and evaluation metrics, which hinders the comparison of different methods and impedes progress in the field [1]. One approach to improving gesture recognition involves using Harris Hawks Optimization (HHO) to fine-tune the parameters of convolutional neural networks (CNNs), resulting in improved recognition performance. However, the high computational complexity of the HHO algorithm makes it difficult to implement in real-time applications, posing challenges for practical use [2]. Another method that has gained attention is the integration of multi-attention mechanisms into CNNs for recognizing hand gestures based on electromyographic (EMG) signals. This technique enhances recognition accuracy and robustness but also increases the complexity of the model, which may hinder real-time performance, especially when working with variable EMG signals [3]. Furthermore, an innovative approach using block-wise neural networks with an auto-learning search framework for sEMG-based finger gesture recognition offers an efficient and accurate recognition system. However, the training time required for this auto-learning search process can be substantial, and the generalization of the model to different users and conditions remains a challenge [4].

For static hand gesture recognition, particularly in sign language, a hybrid feature extraction method combining ORB (Oriented FAST and Rotated BRIEF) descriptors and Gabor filters has been proposed. This method leverages CNNs for efficient feature extraction, improving classification accuracy. However, the high computational demands for feature extraction and the inability to handle unseen gestures or variations in performance limit its scalability [5]. Addressing variations in EMG signals caused by different force levels, especially in gestures performed by amputees, is another key challenge in hand gesture recognition. A deep learning model has been proposed to adapt to these variations, improving the recognition of amputee gestures. While this approach enhances accuracy, the variability in EMG signals and the limited size of the dataset reduce its robustness [6]. To overcome hardware constraints in real-time applications, a quantized CNN model has been designed to achieve real-time hand gesture recognition with reduced computational overhead. Despite its efficiency, quantization can lead to a loss of precision, which may impact recognition accuracy, and balancing performance with efficiency remains a key challenge [7].

In the domain of virtual interactions, one study investigates virtual finger-point reading behaviors by analyzing mouse cursor movements on websites. This analysis reveals user behavior patterns that can inform the design of more accessible and user-friendly websites. However, this approach is limited by its focus on specific user interactions and contexts, which may not apply universally to other applications [8]. A more comprehensive framework combines hybrid-metaheuristic algorithms with deep learning for optimized gesture recognition. While this hybrid approach offers improved accuracy and efficiency, its computational complexity and the need for tuning for different datasets make it less practical in some cases [9]. Similarly, the integration of time-frequency domain analysis with deep learning models for EMG-based gesture recognition in IoT applications promises improved real-time performance. However, challenges related to computational complexity, power consumption, and the integration of IoT devices remain significant barriers to widespread adoption [10]. Transfer learning has also been employed to improve the generalization of CNN-based models for hand gesture recognition using surface EMG signals. This method leverages pre-trained models to reduce training time and improve recognition across different users and conditions. Nonetheless, the effectiveness of transfer learning depends on the similarity between the pre-trained model's domain and the target application, and EMG signal variability remains a limitation [11].

In the realm of 3D hand shape and pose estimation, the HandVoxNet++ voxel-based neural network has been introduced to enhance gesture recognition by capturing volumetric information. While this approach offers improved accuracy, it requires high computational resources, and challenges remain in ensuring accurate estimations across different hand shapes and poses [12]. Additionally, FPGA-based hardware designs for real-time hand gesture recognition provide high efficiency, but the hardware constraints and the need for optimization to balance performance and resource usage are notable limitations in such systems [13]. Augmented reality (AR) systems that support multi-modal interactions, combining gaze, gesture, and speech inputs, have the potential to offer more intuitive and flexible user interactions. However, integration complexity and system latency are significant challenges when ensuring seamless interaction across multiple modalities [14]. Another promising approach involves multi-sensor data fusion for real-time hand gesture tracking in human-computer interfaces. This technique enhances tracking accuracy and responsiveness, but challenges in sensor synchronization and the computational load required for real-time data fusion pose obstacles to its implementation [15].

For embedded systems, temporal convolutional networks (TCNs) are used for robust real-time EMG recognition, particularly in IoT applications. While this approach works well for resource-constrained devices, it faces challenges in achieving the necessary real-time performance and low-latency processing for IoT integration [16]. Furthermore, interactive data transformation in desktop and virtual reality (VR) environments is being explored to provide users with more control and flexibility in data manipulation tasks. However, variability in user experience and the need for VR system optimization to ensure consistent interaction across setups remains a challenge [17]. Skeleton-based action and gesture recognition frameworks, especially for human-robot collaboration, use depth sensors or motion capture systems to facilitate more accurate interactions. Despite their promise, these frameworks are dependent on the quality of skeleton data and can be affected by issues like lighting, occlusions, and dataset diversity, limiting their effectiveness in real-world scenarios [18]. Finally, deep transfer learning models for adaptive gesture recognition using soft e-skin patches have been proposed to reduce the need for extensive training data. While this method offers potential, its effectiveness is influenced by factors such as skin type, environmental conditions, and the suitability of pre-trained models for specific gestures [19]. Similarly, modifications to CNN architectures aim to improve hand gesture estimation, but significant computational resources are still required for real-time processing, and the variability in hand shapes and movements complicates accurate gesture recognition [20].

2.2 Comparative statement (Tabulation) and Research gap Summary

S.No	Paper	Methodology Adopted	Key Observation	Dataset Used	Limitations
1	A Review of the Hand Gesture Recognition System: Current Progress and Future Directions	Comprehensive review of various methods like computer vision, sensor-based, and deep learning techniques.	Advances in gesture recognition systems, need for more robust machine learning techniques.	Various datasets referenced	Lack of standardized datasets and evaluation metrics; challenges in dynamic and complex environments.
2	Hand Gesture Recognition Based on a Harris Hawks Optimized Convolution Neural Network	Uses Harris Hawks Optimization (HHO) algorithm to optimize CNN parameters for gesture recognition.	Improved CNN performance by optimizing parameters using HHO.	EMG data	High computational complexity; challenging for real-time applications.

	3	Electromyographic Hand Gesture Recognition Using Convolutional Neural Network with Multi-Attention	CNN with multi-attention mechanisms to enhance recognition accuracy for EMG-based hand gestures.	Multi-attention improves accuracy and robustness in gesture recognition.	sEMG signals	Increased model complexity; challenges in real-time processing and EMG signal variability.	
	4	Integrated Block-wise Neural Network with Auto-Learning Search Framework for Finger Gesture Recognition	Block-wise neural network with auto-learning search for optimizing network structure and parameters for sEMG-based finger gesture recognition.	Efficient and accurate recognition system for finger gestures.	sEMG signals	High training time; generalization issues across different users and conditions.	

	5	Static Hand Gesture Recognition in Sign Language Based on Convolutional Neural Network with Feature Extraction Method	Combines ORB descriptors and Gabor filters for feature extraction in a CNN-based sign language gesture recognition system.	Effective for static hand gesture recognition, especially in sign language.	Sign language datasets	High computational demands for feature extraction; struggles with unseen gestures and variations in gesture performance.	
	6	Deep Learning Approach to Improve the Recognition of Hand Gesture with Multi Force Variation Using Electromyograph	Deep learning model to address variations in EMG signals from different force levels in gestures, especially for amputees.	Enhanced recognition accuracy for amputee gestures by adapting to force variations.	EMG signals from amputees	EMG signal variability can affect performance; limited dataset size impacts robustness.	

	7	Quantized CNN-based Efficient Hardware Architecture for Real-Time Hand Gesture Recognition	Quantized CNN model designed for efficient hardware implementation to enable real-time gesture recognition.	Real-time performance with reduced computational overhead.	Various gesture datasets	Loss of precision due to quantization; challenges in balancing efficiency and performance.	
	8	Virtual Finger-Point Reading Behaviors: A Case Study of Mouse Cursor Movements on a Website	Analyzes mouse cursor movements to understand virtual finger-point reading behaviors for web design.	Provides insights into user behavior that can inform website design and accessibility.	Website interaction data	Limited to specific user interactions and contexts; may not generalize across different websites or applications.	

	9	A Comprehensive Framework for Hand Gesture Recognition Using Hybrid-Metaheuristic Algorithms and Deep Learning Models	Combines hybrid-metaheuristic algorithms with deep learning for optimized gesture recognition.	Improved accuracy and efficiency in gesture recognition through a hybrid approach.	Multiple gesture datasets	Computationally intensive and complex; requires tuning and adaptation for different datasets and applications.	
	10	Automated Recognition of Hand Gestures From Multichannel EMG Sensor Data Using Time–Frequency Domain Deep Learning for IoT Applications	Uses time-frequency domain analysis combined with deep learning for real-time hand gesture recognition from multichannel EMG data.	Suitable for IoT applications, enhances recognition accuracy using time-frequency domain deep learning.	Multichannel EMG data	Computational complexity and power consumption challenges for real-time IoT integration.	

	11	Hand Gesture Recognition Based on Surface Electromyograph Using Convolutional Neural Network with Transfer Learning Method	Uses CNN and transfer learning to recognize hand gestures from surface EMG signals.	Improved generalization across users and conditions using transfer learning.	Surface EMG signals	Transfer learning effectiveness depends on domain similarity; variability in EMG signals can affect performance.	
	12	HandVoxNet++: 3D Hand Shape and Pose Estimation Using Voxel-Based Neural Networks	Voxel-based neural network for accurate 3D hand shape and pose estimation to enhance gesture recognition.	Improved hand shape and pose estimation accuracy for gesture recognition.	3D hand shape datasets	High computational requirements; challenges in accurate estimation across different hand shapes and poses.	

	13	Hardware Architecture Design for Hand Gesture Recognition System on FPGA	FPGA-based hardware design for real-time hand gesture recognition.	High efficiency and real-time performance achieved with FPGA architecture.	Various gesture datasets	Hardware constraints; requires significant optimization to balance performance and resource usage.	
	14	Interaction with Gaze, Gesture, and Speech in a Flexibly Configurable Augmented Reality System	Multi-modal interaction with gaze, gesture, and speech in an augmented reality (AR) system.	Potential for more intuitive user interactions across various scenarios in AR.	AR system interaction data	Integration complexity and system latency; challenges in ensuring seamless interaction across multiple modalities.	

	15	Real-Time Hand Gesture Tracking for Human–Computer Interface Based on Multi-Sensor Data Fusion	Multi-sensor data fusion for real-time hand gesture tracking in human-computer interfaces.	Enhances accuracy and responsiveness in gesture tracking for interfaces.	Various sensor data	Challenges in sensor synchronization and real-time processing; requires significant computational resources for data fusion.	
	16	Robust Real-Time Embedded EMG Recognition Framework Using Temporal Convolutional Networks on a Multicore IoT Processor	Uses Temporal Convolutional Networks for real-time EMG recognition in embedded systems, suitable for IoT applications.	Robust real-time EMG recognition for resource-constrained IoT devices.	EMG data	Real-time performance challenges; integration with IoT devices for low-latency processing.	

	17	Interactive Data Transformation on Desktop and in Virtual Reality	Demonstrates interactive data transformation in desktop and VR environments for more user control and flexibility in data manipulation tasks.	Enhances user interaction with data across desktop and VR environments.	User interaction data	Variability in user experience; VR system requirements for consistent interaction across setups.	
	18	A General Skeleton-Based Action and Gesture Recognition Framework for Human–Robot Collaboration	Skeleton-based action and gesture recognition framework for human-robot collaboration using depth sensors or motion capture systems.	Facilitates human-robot collaboration with accurate gesture recognition.	Annotated skeleton data	Performance dependent on skeleton data quality; affected by lighting, occlusions, and dataset diversity.	

19	Deep Transfer Learning-Based Adaptive Gesture Recognition of a Soft E-Skin Patch with Reduced Training Data	Uses deep transfer learning for adaptive gesture recognition with minimal training data via a soft e-skin patch.	Reduces need for extensive training data and time for gesture recognition.	Soft e-skin patch gesture data	Skin type and environmental conditions affect performance; pre-trained models may not always be suitable.
20	An Accurate Estimation of Hand Gestures Using Optimal Modified Convolutional Neural Network	Modified CNN architecture to improve hand gesture estimation for better model performance.	Improved hand gesture estimation accuracy using modified CNN.	Depth cameras, motion capture systems	Requires significant computational resources; accuracy affected by hand shape, size, and movement variability.

Research Gap Summary: Hand Gesture Recognition

The field of hand gesture recognition has witnessed significant advancements in accuracy, robustness, and adaptability. However, several critical research gaps remain that hinder the effectiveness and applicability of these systems in real-world scenarios:

1. Robustness in Complex Environments: Existing systems still struggle to perform reliably in dynamic and uncontrolled settings. There is a pressing need for the development of adaptable algorithms that can maintain consistent performance across various environments.
2. Standardization of Datasets and Metrics: The absence of standardized datasets and evaluation metrics obstructs meaningful comparisons between models. This gap limits the generalizability of findings and hinders the ability to draw broad conclusions applicable to real-world applications.
3. Real-Time Processing: High computational demands associated with optimization algorithms, attention mechanisms, and deep learning models significantly impede real-time processing capabilities. This

limitation is particularly critical for practical applications such as AI-based virtual mice or gesture-controlled interfaces in virtual and augmented reality (VR/AR).

4. Generalization Across Users and Contexts: Variability in electromyography (EMG) signals, force levels, and individual user conditions leads to inconsistent model performance across different users, gesture types, and environmental contexts. Improved methods for enhancing generalization are essential.

5. Hardware Efficiency and Power Consumption: Real-time performance in hardware-implemented systems, including those utilizing FPGA, often requires compromises between power consumption and processing speed. There is a need for efficient designs that strike a better balance between these competing factors.

6. Handling Unseen and Variational Gestures: Many models demonstrate limitations in recognizing unseen gestures or subtle variations, particularly in areas like sign language recognition. Developing more adaptive systems that can handle such variability is crucial.

7. Integration Complexity in Multi-modal Systems: Systems that integrate multiple modalities—such as gesture, gaze, and speech—face challenges related to seamless integration and maintaining low-latency operations. Addressing these integration complexities is vital for the effectiveness of multi-modal gesture recognition systems.

8. Data Quality and Variability: Factors such as lighting conditions, sensor noise, and individual differences in physical characteristics (e.g., skin type, hand shape) can adversely affect data quality. These variations complicate the creation of universally effective gesture recognition systems.

2.3 Hardware Requirements

1. Camera: High-resolution camera (e.g., 1080p) with a good frame rate (30fps or higher) for accurate finger tracking.
2. Computer: A computer with a multi-core processor (e.g., Intel i5 or higher) and sufficient RAM (8GB or more) to process the video stream and run the AI models.
3. GPU (Optional): A dedicated GPU (e.g., NVIDIA GTX series) for faster AI processing and rendering.

2.4 Software Requirements

1. Operating System:

The development of finger tracking and AI algorithms can be done on a variety of operating systems based on your preferences and requirements:

- Windows: A commonly used operating system for general development, providing strong support for various software tools and libraries.
- macOS: Preferred for its Unix-based environment, which supports a wide range of development tools, especially for design and high-performance computing tasks.
- Linux: Highly recommended for AI and computer vision applications due to its stability, open-source nature, and extensive community support. It is widely used in server-side development and AI model deployment.

2. Programming Languages:

The following programming languages are most suitable for developing finger tracking and AI algorithms:

- Python: A popular choice for AI and computer vision development, Python is widely used for machine learning tasks due to its simplicity and large ecosystem of libraries such as OpenCV, TensorFlow, and PyTorch.
- C++: Known for its performance, C++ is preferred for real-time applications that require low latency and high-speed execution, especially in hardware-optimized tasks like gesture recognition.

3. Libraries and Frameworks:
To implement the necessary functionality for finger tracking and AI, the following libraries and frameworks are essential:

- OpenCV: A powerful library used for computer vision and image processing tasks. OpenCV helps in capturing video input, detecting features, tracking hand movements, and performing background subtraction.
- TensorFlow / PyTorch: These frameworks are used to develop and deploy AI models. TensorFlow and PyTorch support deep learning models such as CNNs (Convolutional Neural Networks), which are ideal for hand gesture and finger tracking tasks. Both frameworks offer flexibility, performance, and scalability.
- NumPy / SciPy: These libraries are indispensable for numerical computations, handling large datasets, performing matrix operations, and solving optimization problems. NumPy provides the core data structure for efficient data handling, while SciPy offers advanced scientific algorithms.

4. Development Tools:
To streamline the development process, the following tools are commonly used:

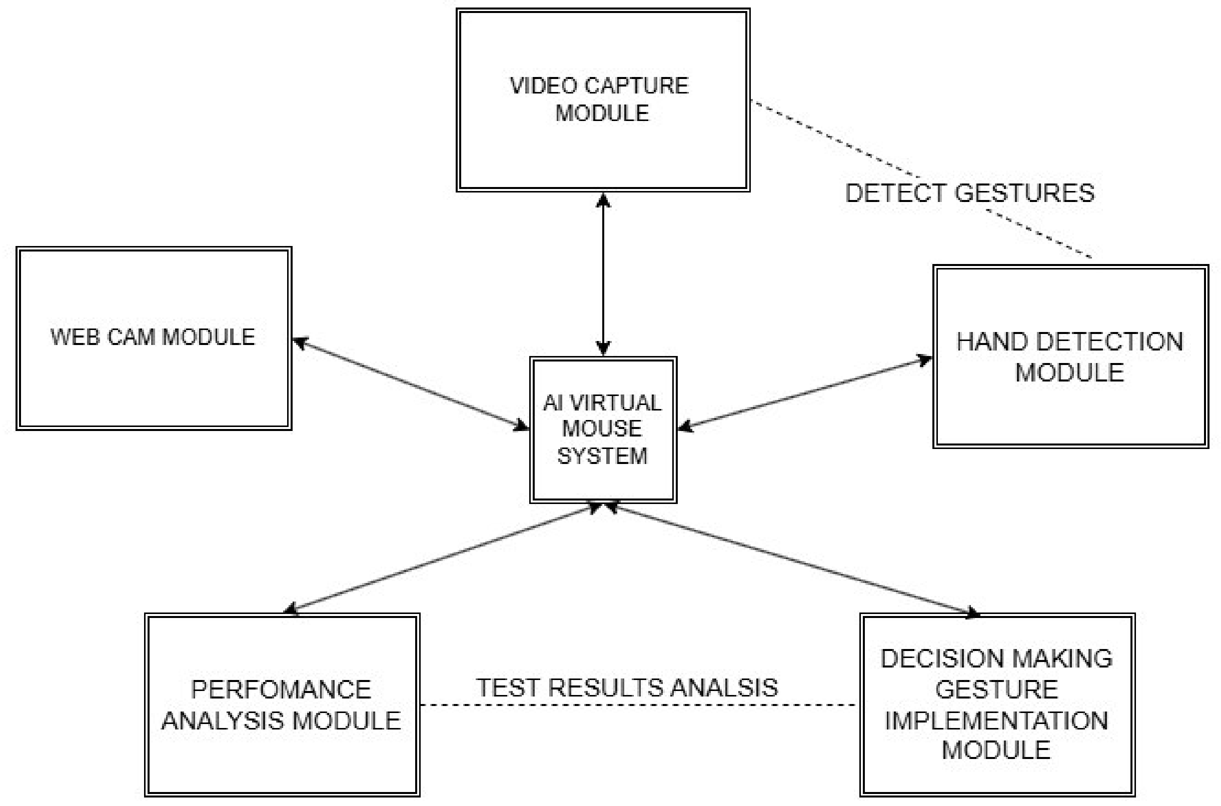
- Integrated Development Environment (IDE):
  PyCharm: A feature-rich IDE specifically designed for Python development. PyCharm supports code completion, debugging, and integrates well with libraries such as TensorFlow, PyTorch, and OpenCV.
  Visual Studio Code (VS Code): A lightweight and customizable IDE that supports multiple languages, including Python and C++. It also has an extensive library of extensions that can help with coding, debugging, and project management in AI and computer vision tasks.
- Version Control System:
  Git: Git is essential for managing source code and collaboration. It allows developers to track changes, create branches for feature development, and merge code. Platforms such as GitHub, GitLab, or Bitbucket provide cloud repositories that facilitate version control and teamwork in software development.

5. Additional Software:
To ensure proper configuration and optimize the system's performance, the following software may be required:

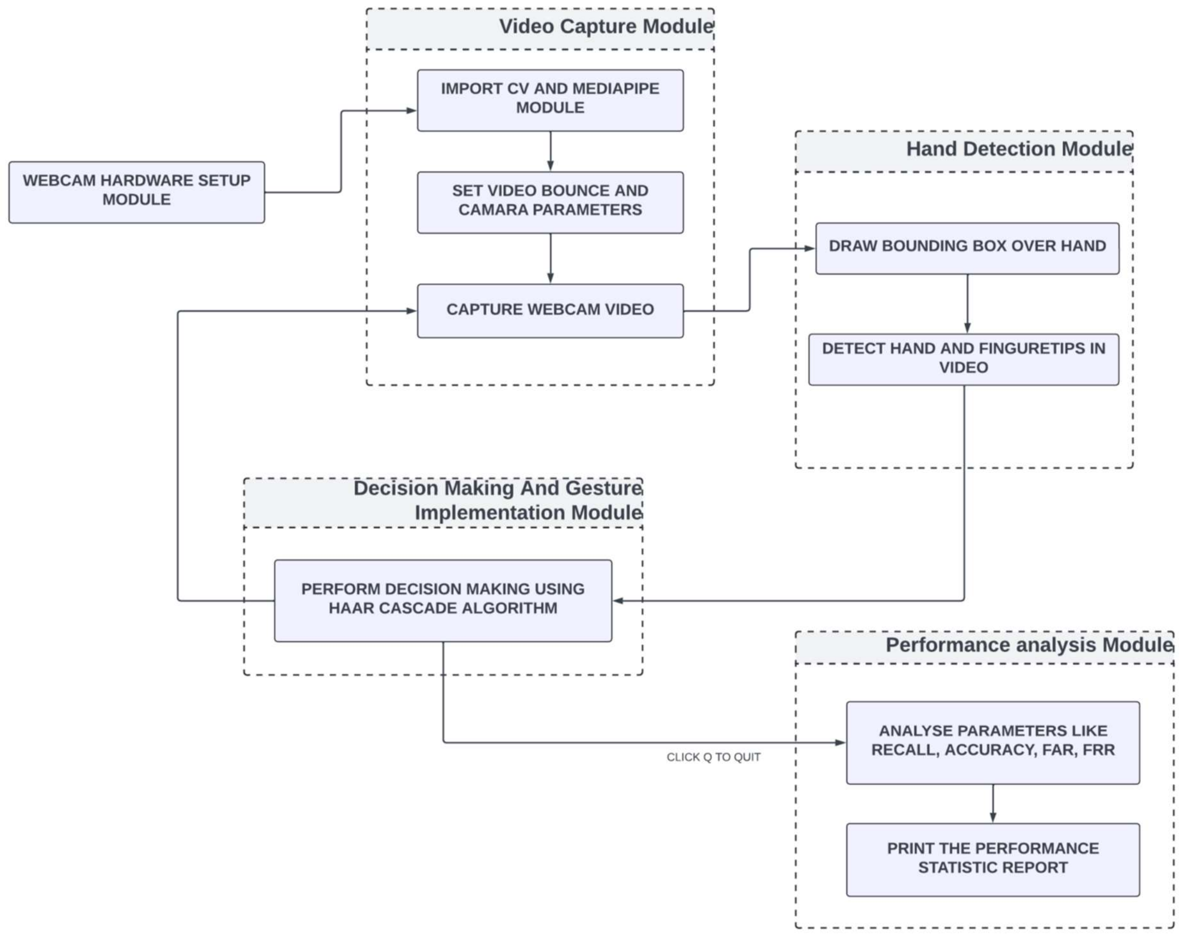
- Software for Camera Calibration and Configuration:
  Proper camera calibration is essential for achieving accurate finger tracking. Calibration software helps adjust camera settings such as focus, exposure, and resolution to ensure high-quality video input. OpenCV provides built-in tools for camera calibration, enabling developers to set up cameras and fine-tune the system's capture parameters.
- User Interface Software:
  A user-friendly interface is necessary to enable users to interact with the system, configure settings, and monitor performance. Software like Tkinter (for Python) or Qt (for C++) can be used to create custom GUIs (Graphical User Interfaces) that allow users to modify parameters such as sensitivity, tracking speed, or model performance, and view real-time results.

3. System Design

3.1 High-Level Design



3.2 Low-Level Design (Detailed design)


Video Capture Module
IMPORT CV AND MEDIAPIPE MODULE
SET VIDEO BOUNCE AND CAMARA PARAMETERS
CAPTURE WEBCAM VIDEO
WEBCAM HARDWARE SETUP MODULE
Hand Detection Module
DRAW BOUNDING BOX OVER HAND
DETECT HAND AND FINGURETIPS IN VIDEO
Decision Making And Gesture Implementation Module
PERFORM DECISION MAKING USING HAAR CASCADE ALGORITHM
CLICK Q TO QUIT
Performance analysis Module
ANALYSE PARAMETERS LIKE RECALL, ACCURACY, FAR, FRR
PRINT THE PERFORMANCE STATISTIC REPORT

Hardware and Software Specification:

Hardware: High-resolution Camera (1080p, 30fps or higher).

Computer with a multi-core processor (Intel i5 or higher) and 8GB RAM.

Optional: Dedicated GPU (NVIDIA GTX series) for faster processing.

Software: Operating System: Windows, macOS, or Linux.

Programming Language: Python.

Libraries and Frameworks: OpenCV (for image processing), MediaPipe (hand detection), PyAutoGUI and Pynput (for mouse control), Numpy (numerical calculations).

Modules of the Project:

Input Module: Captures video feed from the camera and passes it to the processing units.

Hand Tracking Module: Uses MediaPipe to detect hands and identify key landmarks.

Gesture Recognition Module: Analyzes hand positions to recognize gestures.

Cursor Control Module: Maps gestures to cursor actions (move, click, scroll).

Feedback Module: Provides visual feedback on recognized gestures.

Detailed Description of the Dataset:

The project does not use a pre-collected dataset. Instead, it processes real-time video input directly from the camera to detect hand gestures dynamically. This approach eliminates the need for a static dataset and adapts to live input conditions.

3.3 Methodology

Methodologies Used:

- Hand Detection and Tracking (MediaPipe): Detects hands and extracts landmarks using machine learning models optimized for real-time performance.
- Gesture Recognition: Utilizes geometric analysis of detected landmarks (angles between points) to determine specific gestures.
- Action Mapping: Maps recognized gestures to specific cursor actions using PyAutoGUI, enabling interaction without physical contact.
- Smoothing Algorithms: Implements smoothing techniques to reduce jitter in cursor movements, enhancing user experience.

4. Results and Discussion

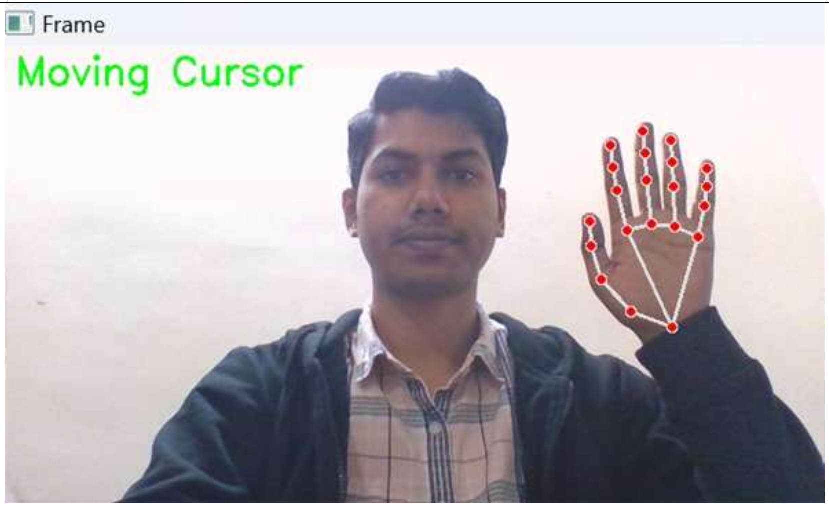
4.1 Implementation Code and Results

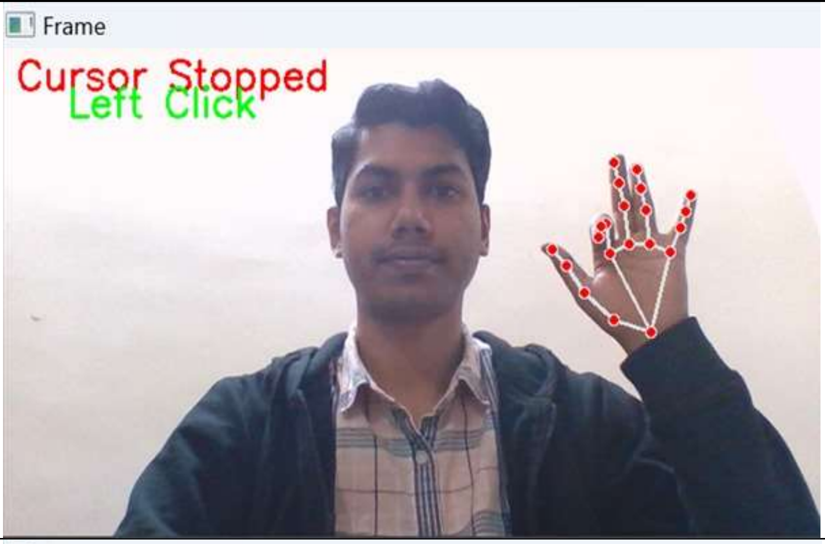
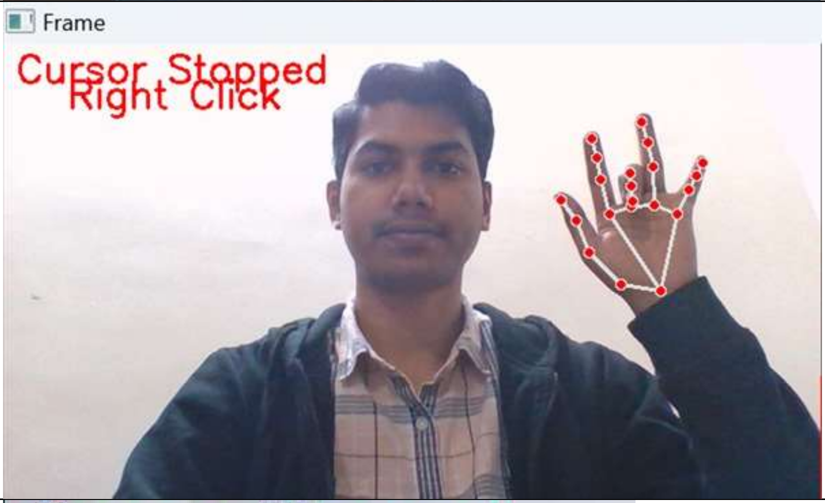
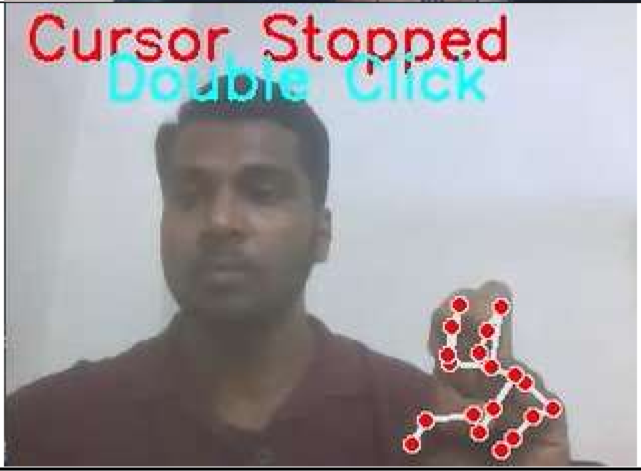
Code:

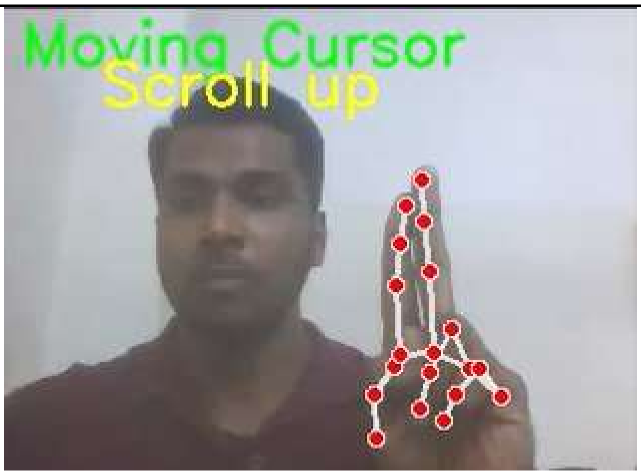
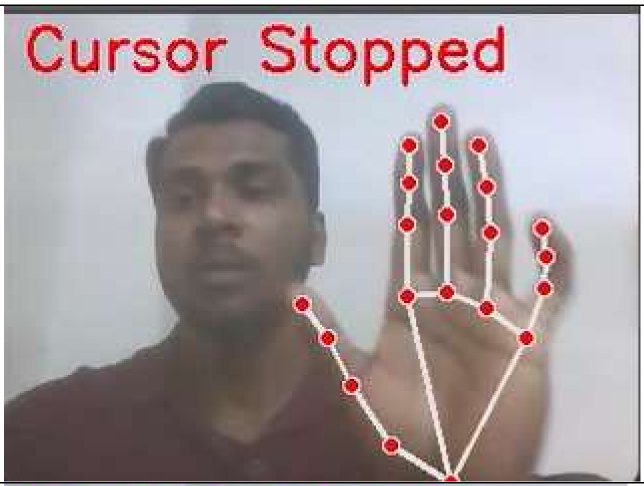
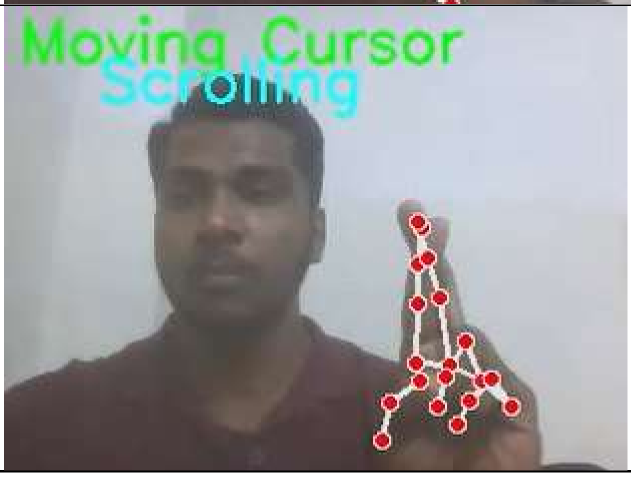
```
import cv2
import mediapipe as mp
import pyautogui
import random
import util
from pynput.mouse import Button, Controller
import numpy as np
from new import test
import ctypes

test()

# Initialize the mouse controller
mouse = Controller()

# Get screen dimensions
screen_width, screen_height = pyautogui.size()

# Initialize Mediapipe Hands module
mpHands = mp.solutions.hands
hands = mpHands.Hands(
    static_image_mode=False,
    model_complexity=0,
    min_detection_confidence=0.7,
    min_tracking_confidence=0.7,
    max_num_hands=1
)

# Variables for smoothing cursor movement
prev_x, prev_y = 0, 0
smoothing_factor = 1

def find_finger_tip(processed):
    """Finds the index finger tip coordinates."""
    if processed.multi_hand_landmarks:
        hand_landmarks = processed.multi_hand_landmarks[0]
        index_finger_tip = hand_landmarks.landmark[mpHands.HandLandmark.INDEX_FINGER_TIP]
        return index_finger_tip
    return None

def move_mouse(index_finger_tip):
    """Smoothly moves the mouse cursor based on the index finger tip position."""
    global prev_x, prev_y
    if index_finger_tip is not None:
        target_x = np.interp(index_finger_tip.x, [0, 1], [0, screen_width])
        target_y = np.interp(index_finger_tip.y, [0, 1], [0, screen_height])
        current_x = prev_x + (target_x - prev_x) / smoothing_factor
        current_y = prev_y + (target_y - prev_y) / smoothing_factor
        pyautogui.moveTo(current_x, current_y)
        prev_x, prev_y = current_x, current_y

def is_thumb_closed(landmark_list):
    """Checks if the thumb is closed."""
    thumb_tip = landmark_list[4]
    index_mcp = landmark_list[5]
    thumb_index_distance = np.linalg.norm(np.array(thumb_tip) - np.array(index_mcp))
    threshold = 0.06
    return thumb_index_distance < threshold

def is_left_click(landmark_list, thumb_index_dist):
    """Checks if the left-click gesture is performed."""
    index_angle = util.get_angle(landmark_list[5], landmark_list[6], landmark_list[8])
    middle_angle = util.get_angle(landmark_list[9], landmark_list[10], landmark_list[12])
    return index_angle < 50 and middle_angle > 100 and thumb_index_dist > 60

def is_right_click(landmark_list, thumb_index_dist):
    """Checks if the right-click gesture is performed."""
    middle_angle = util.get_angle(landmark_list[9], landmark_list[10], landmark_list[12])
    index_angle = util.get_angle(landmark_list[5], landmark_list[6], landmark_list[8])
    return middle_angle < 50 and index_angle > 100 and thumb_index_dist > 60

def is_double_click(landmark_list, thumb_index_dist):
    """Checks if the double-click gesture is performed."""
    index_angle = util.get_angle(landmark_list[5], landmark_list[6], landmark_list[8])
    middle_angle = util.get_angle(landmark_list[9], landmark_list[10], landmark_list[12])
    return index_angle < 50 and middle_angle < 50 and thumb_index_dist > 60

def is_screenshot(landmark_list, thumb_index_dist):
    """Checks if the screenshot gesture is performed."""
    index_angle = util.get_angle(landmark_list[5], landmark_list[6], landmark_list[8])
    middle_angle = util.get_angle(landmark_list[9], landmark_list[10], landmark_list[12])
    return index_angle < 50 and middle_angle < 50 and thumb_index_dist < 40
```

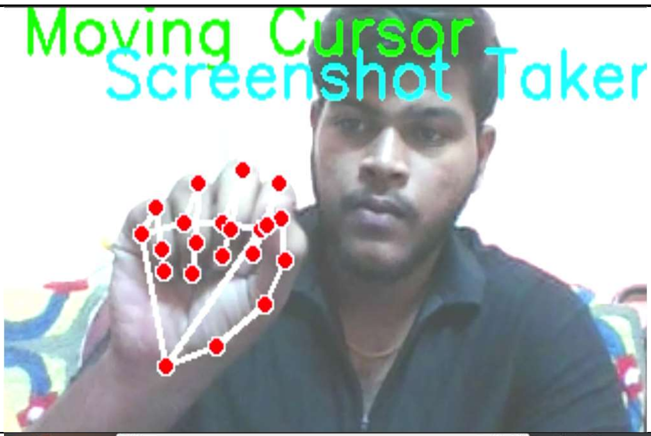
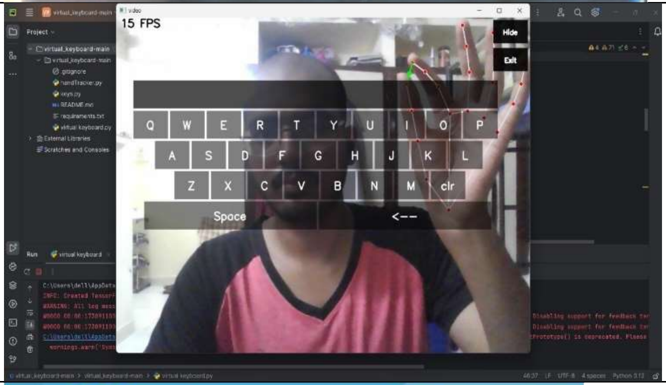
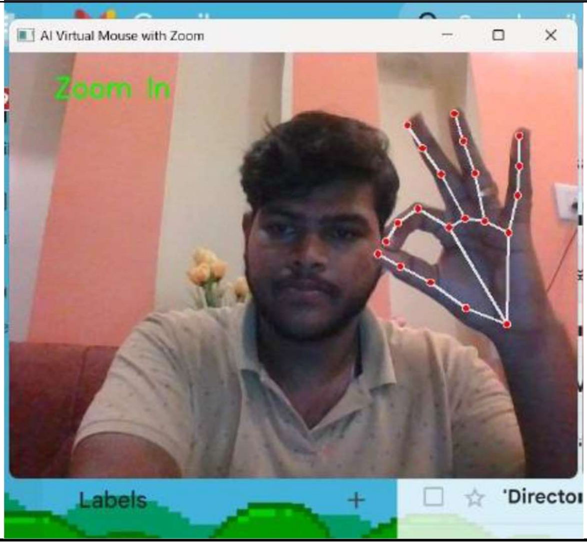
```
def is_two_finger_scroll(landmark_list):
    """Checks if the two-finger scroll gesture is performed."""
    index_tip = landmark_list[8]
    middle_tip = landmark_list[12]
    distance = np.linalg.norm(np.array(index_tip) - np.array(middle_tip))
    return distance < 0.05  # Adjust the distance threshold for better recognition

def is_palm_tilt_scroll(landmark_list):
    """Checks if the palm tilt scroll gesture is performed by checking wrist tilt."""
    wrist_angle = util.get_angle(landmark_list[0], landmark_list[1], landmark_list[2])
    return wrist_angle < 150 or wrist_angle > 210  # Adjust angle range for tilt

def is_hand_rotation(landmark_list):
    """Checks if the hand is rotated clockwise or counterclockwise."""
    # Get angle between wrist and base of fingers
    wrist_angle = util.get_angle(landmark_list[0], landmark_list[1], landmark_list[2])
    # Define a threshold for rotation detection
    return wrist_angle < 50 or wrist_angle > 330  # Adjust angle range for rotation detection

def scroll_page(direction):
    """Scrolls the page up or down based on the direction."""
    if direction == 'up':
        pyautogui.scroll(300)
    elif direction == 'down':
        pyautogui.scroll(-300)

def adjust_volume(direction):
    """Adjusts the system volume based on the direction."""
    if direction == 'up':
        ctypes.windll.user32.SendMessageW(0xFFFF, 0x319, 0x3019, 0x0000)
    elif direction == 'down':
        ctypes.windll.user32.SendMessageW(0xFFFF, 0x319, 0x301A, 0x0000)

def detect_gesture(frame, landmark_list, processed):
    """Detects gestures and performs corresponding actions."""
    if len(landmark_list) >= 21:
        index_finger_tip = find_finger_tip(processed)
        thumb_index_dist = util.get_distance([landmark_list[4], landmark_list[5]])

        if is_thumb_closed(landmark_list):
            move_mouse(index_finger_tip)
            cv2.putText(frame, "Moving Cursor", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        else:
            cv2.putText(frame, "Cursor Stopped", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

        if is_left_click(landmark_list, thumb_index_dist):
            mouse.click(Button.left, 1)
            cv2.putText(frame, "Left Click", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        elif is_right_click(landmark_list, thumb_index_dist):
            mouse.click(Button.right, 1)
            cv2.putText(frame, "Right Click", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
        elif is_double_click(landmark_list, thumb_index_dist):
            pyautogui.doubleClick()
            cv2.putText(frame, "Double Click", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 2)
        elif is_screenshot(landmark_list, thumb_index_dist):
            im1 = pyautogui.screenshot()
            label = random.randint(1, 1000)
            im1.save(f'my_screenshot_{label}.png')
            cv2.putText(frame, "Screenshot Taken", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0),
2)
        elif is_two_finger_scroll(landmark_list):
            pyautogui.scroll(-100)
            cv2.putText(frame, "Scrolling", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 2)
        elif is_palm_tilt_scroll(landmark_list):
            tilt_direction = 'up' if landmark_list[2][1] < landmark_list[0][1] else 'down'
            scroll_page(tilt_direction)
            cv2.putText(frame, f"Scroll {tilt_direction}", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
255), 2)
        elif is_hand_rotation(landmark_list):
            rotation_direction = 'up' if landmark_list[2][0] < landmark_list[1][0] else 'down'
            adjust_volume(rotation_direction)
            cv2.putText(frame, f"Volume {rotation_direction.capitalize()}", (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 255), 2)

def main():
    draw = mp.solutions.drawing_utils
    cap = cv2.VideoCapture(0)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)

    try:
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break

            frame = cv2.flip(frame, 1)
            frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            processed = hands.process(frameRGB)
            landmark_list = []

            if processed.multi_hand_landmarks:
                hand_landmarks = processed.multi_hand_landmarks[0]
                draw.draw_landmarks(frame, hand_landmarks, mpHands.HAND_CONNECTIONS)
                for lm in hand_landmarks.landmark:
                    landmark_list.append((lm.x, lm.y))
            detect_gesture(frame, landmark_list, processed)
            cv2.imshow('Frame', frame)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

    finally:
        cap.release()
        cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```

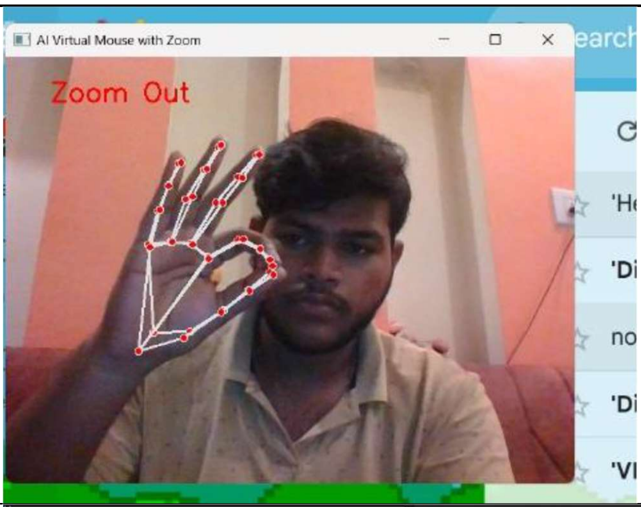
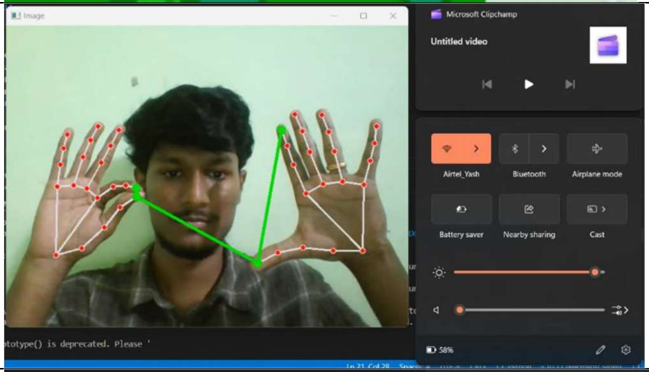
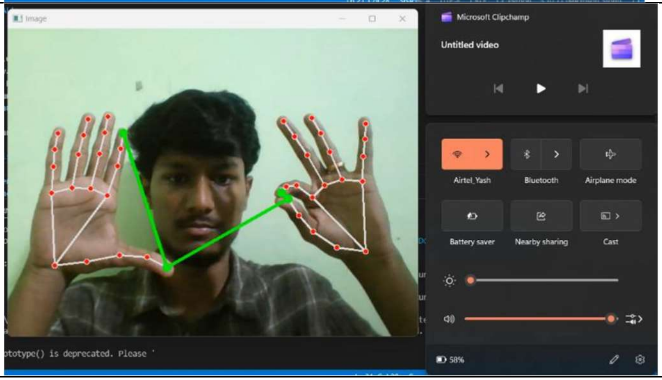
RESULTS:

S.NO	GESTURE NAME	DESCRIPTION (HOW IT WORKS)	SCREENSHOTS
1.	Moving Cursor	Moves the mouse cursor based on the position of the index fingertip. Also, with the full hand extended and closing the thumb finger To perform: Point the index finger straight while closing the thumb.	

2.	Left Click	Performs a left mouse click. To perform: close your index finger bent, and the rest of the fingers opened, resembling a pinching motion.	 Frame Cursor Stopped Left Click
3.	Right Click	Performs a right mouse click. To perform: Bend your middle finger down with the thumb open, while keeping the index finger slightly bent.	 Frame Cursor Stopped Right Click
4.	Double Click	Performs a double mouse click. To perform: Make a pinching motion with both the index and middle fingers bent, keeping the thumb open.	 Cursor Stopped Double Click

5.	Scroll up	Extend the two fingers-> index and the middle fingers and close all the other fingers completely	
6.	Cursor Stopped	Held hand open and completely still to stop the cursor	
6.	Scroll down	Cross the two fingers index and middle finger to scroll down.	

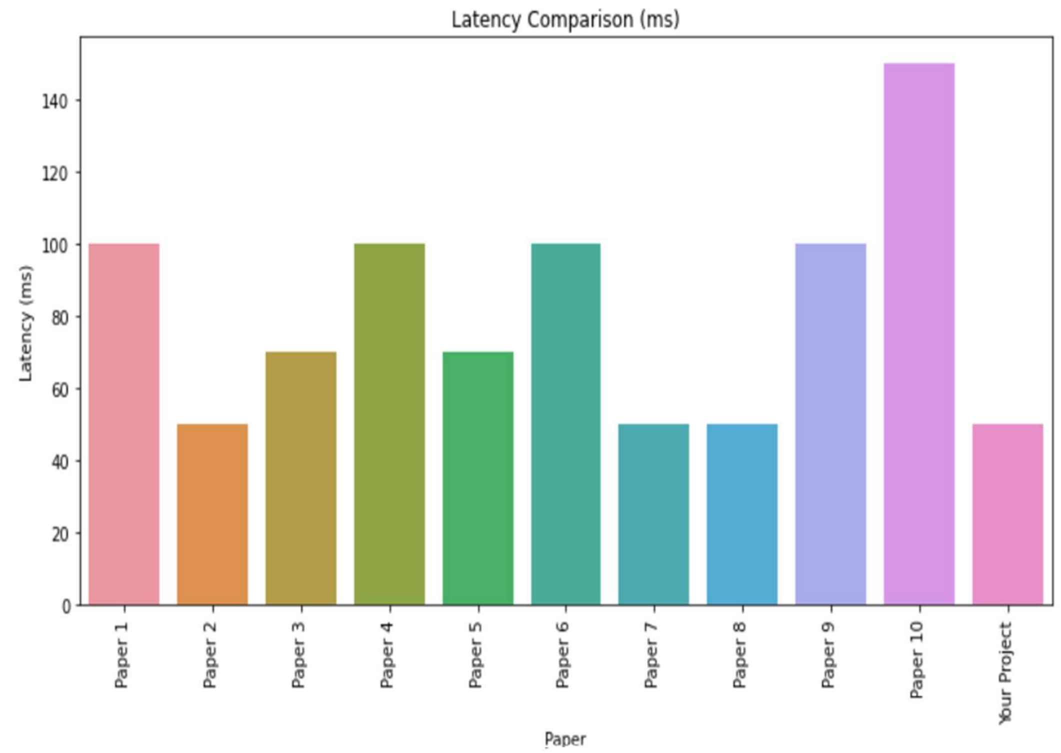
7.	screenshot	Fold all the five fingers at a time to make a screenshot on the spot	
8.	Virtual keyboard	Move your hand near show icon and bring together your thumb and index finger to select keys on virtual keyboard	 15 FPS Hide Exit Q W E R T Y U I O P A S D F G H J K L Z X C V B N M clr Space <--
9.	Zoom IN	Move your right hand's thumb and index finger closely like (double tap) for twice (0.3 sec) to zoom in	 AI Virtual Mouse with Zoom Zoom In Labels 'Director

10.	Zoom OUT	Move your left hand's index finger and thumb finger closer twice (double tapping sign) to zoom in	
11.	Volume Control	Move the thumb and index fingers of the left hand closer and further to control the volume.	 Image Microsoft Clipchamp Untitled video Airtel_Yash Bluetooth Airplane mode Battery saver Nearby sharing Cast 58% totype() is deprecated. Please '
12.	Brightness Control	Move the thumb and index fingers of the right hand closer and further to control the volume.	 Image Microsoft Clipchamp Untitled video Airtel_Yash Bluetooth Airplane mode Battery saver Nearby sharing Cast 58% totype() is deprecated. Please '

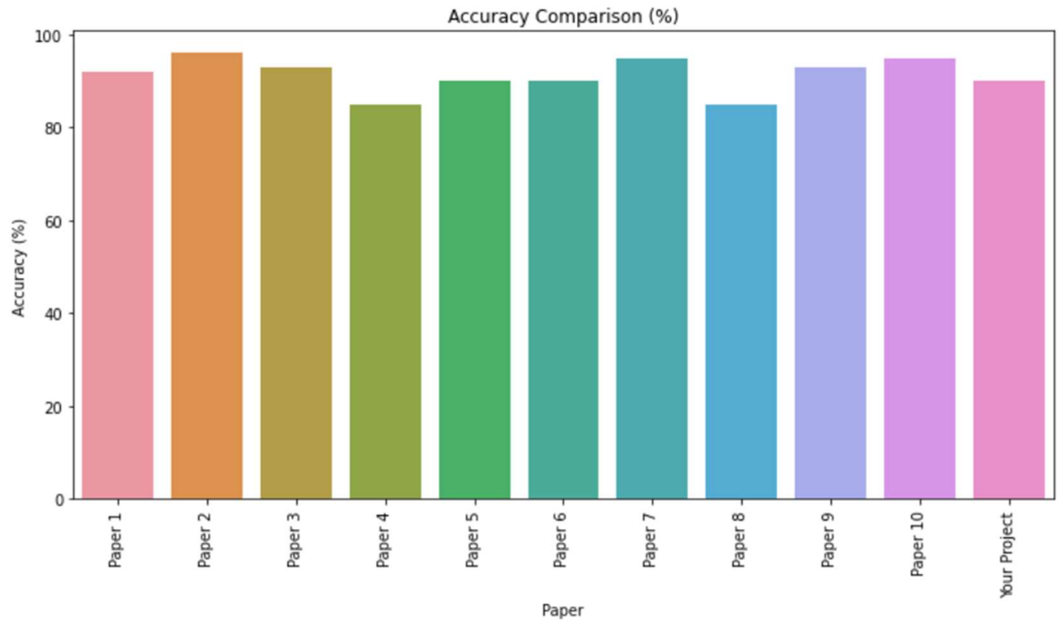
4.2 Metrics
Funtional requirements are assessed based on Latency (ms), Computational Complexity (FLOPS), Accuracy (%), Power Consumption
Non functional requirement are assessed based on Adaptability and Real-World Feasibility

4.3 Results in table/Graph/Data (No screenshots, only text form of data in table), Graph should be drawn using Excel or python

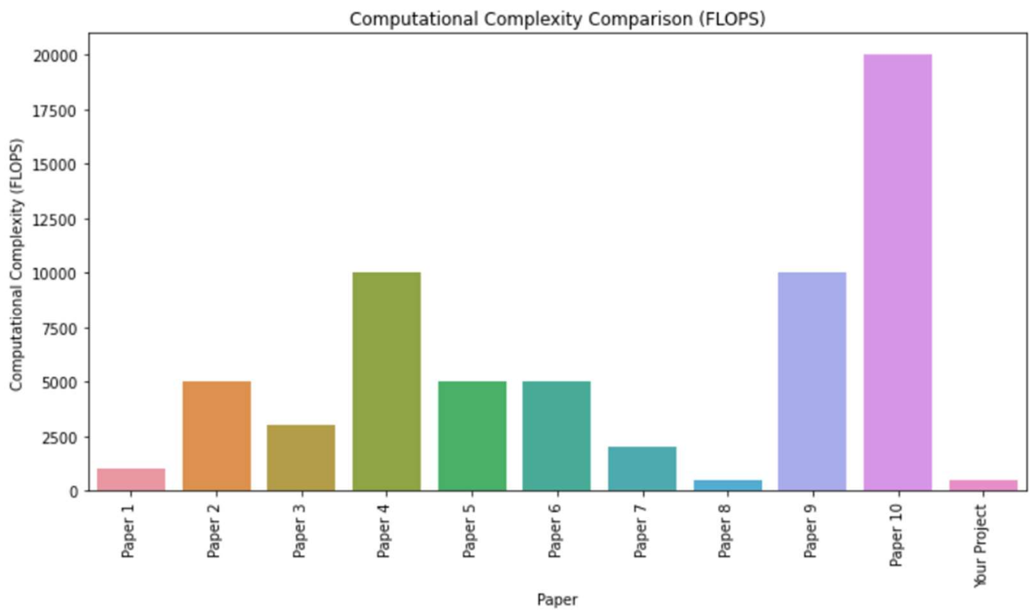
Paper/Project	Latency (ms)	Computational Complexity (FLOPS)	Accuracy (%)	Power Consumption	Adaptability	Real-World Feasibility
Virtual Mouse (Hand Gesture Control via Webcam)	Low (~30-50)	Low (Webcam + Simple Model)	90-95	Low (Webcam)	High	Very High

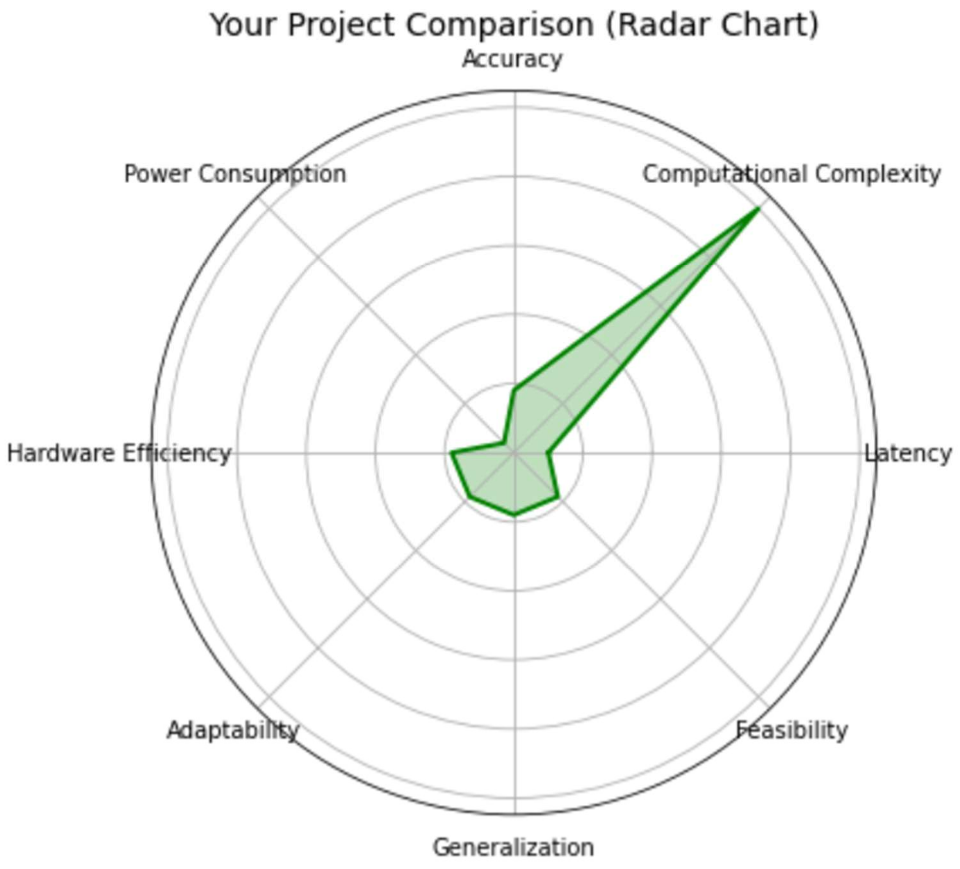


Latency Comparison (ms)
Latency (ms)
0
20
40
60
80
100
120
140
Paper 1
Paper 2
Paper 3
Paper 4
Paper 5
Paper 6
Paper 7
Paper 8
Paper 9
Paper 10
Your Project
Paper



Accuracy Comparison (%)
Accuracy (%)
0
20
40
60
80
100
Paper 1
Paper 2
Paper 3
Paper 4
Paper 5
Paper 6
Paper 7
Paper 8
Paper 9
Paper 10
Your Project
Paper



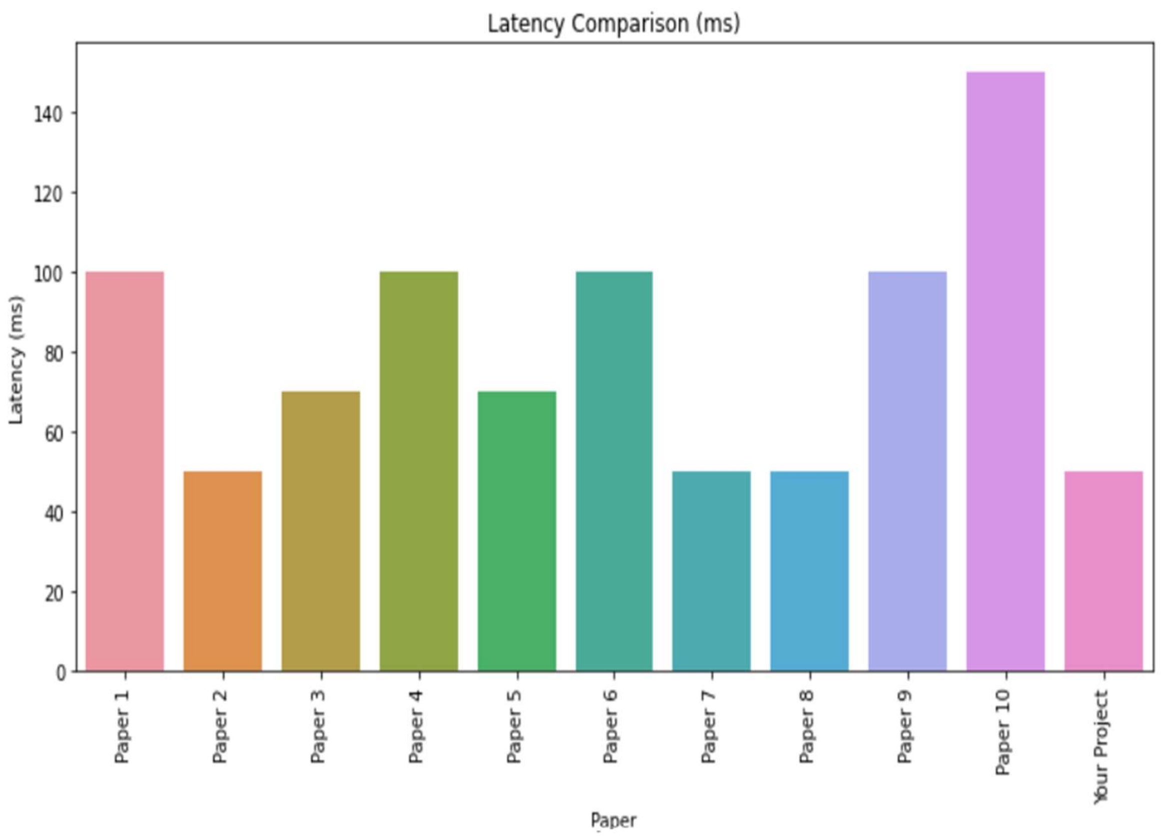


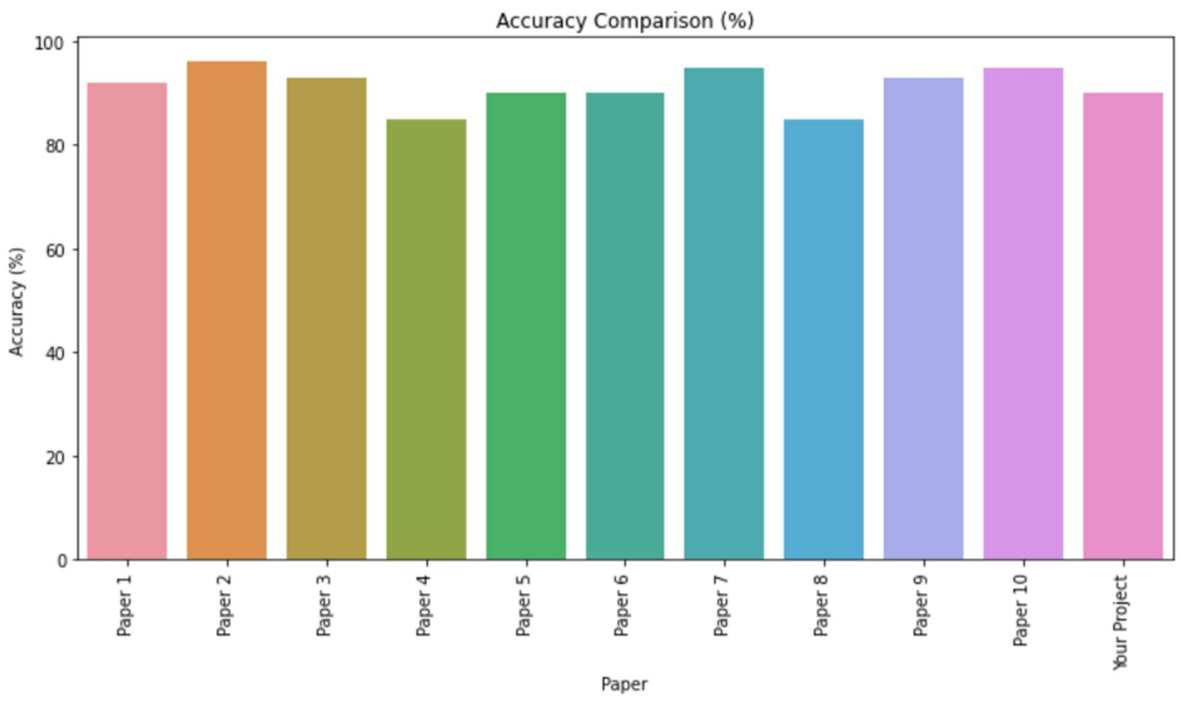
4.4 Mapping the results with problem statement and existing systems

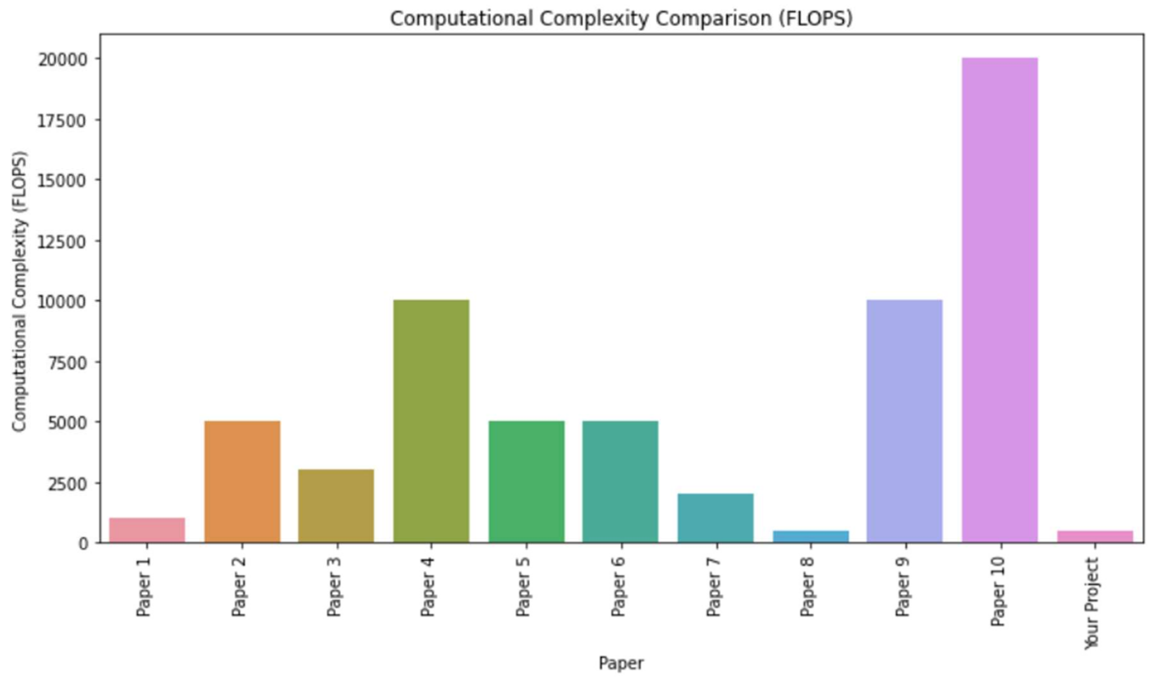
Comparison of your project with an existing system

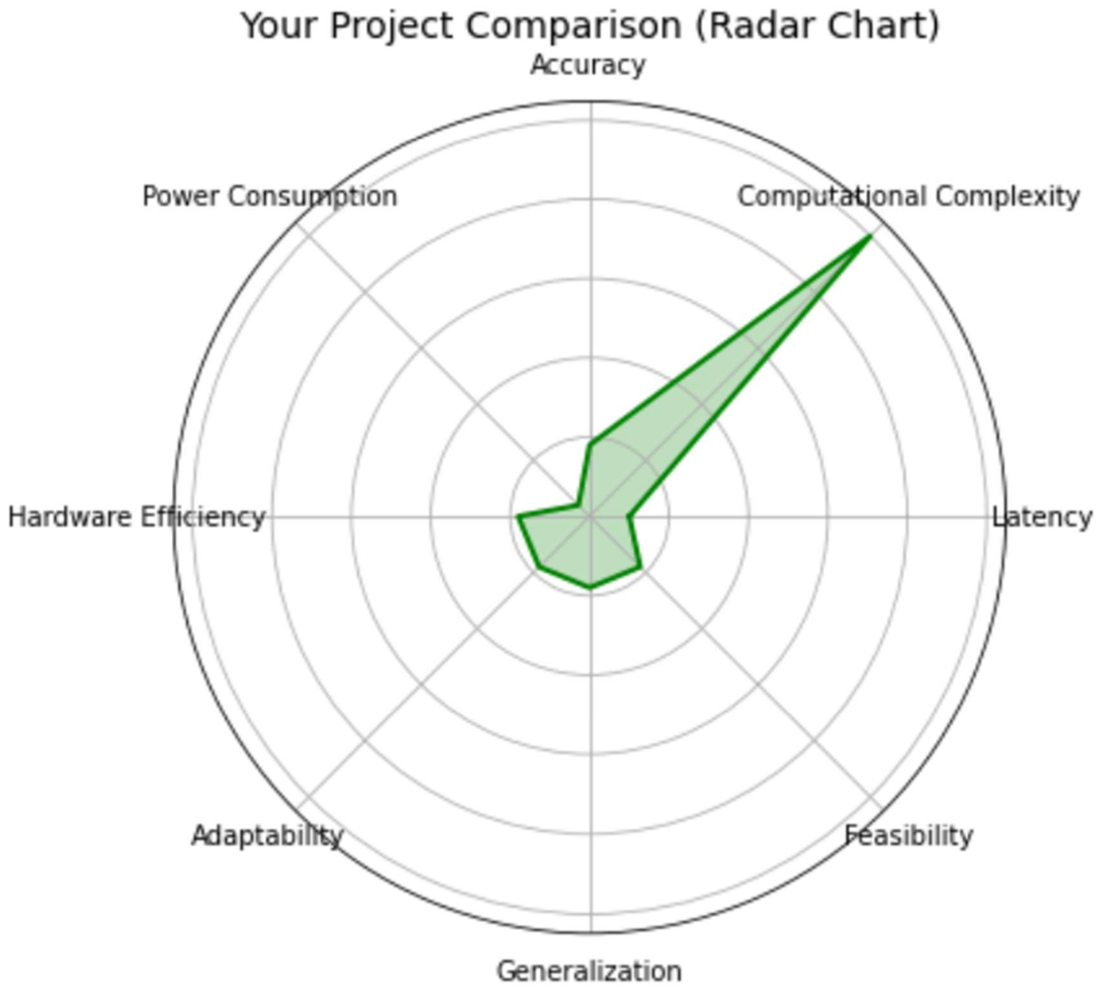
a. Graphs and a supportive table of values required:

S.No	Paper/Project	Latency (ms)	Computational Complexity (FLOPS)	Accuracy (%)	Power Consumption	Adaptability	Real-World Feasibility
[1]	A Review of Hand Gesture Recognition Systems	High (~100-200)	High (Various Methods)	85-95	High (Standard Cameras)	Medium	Medium
[2]	Hand Gesture Recognition Using Harris Hawks Optimized CNN	Very High (~50)	Very High (Optimized CNN)	95+	High (GPU)	Low	Low
[3]	Electromyographic Gesture Recognition with Multi-Attention CNN	Medium (~70-100)	High (CNN-Based)	90-95	High (EMG Sensors)	Medium	Medium
[4]	Block-wise Neural Network with Auto-learning Search	High (~100-200)	Very High (Auto-Learning Search)	80-90	Very High (EMG Sensors)	Low	Low
[5]	Sign Language Gesture Recognition (ORB & Gabor Filters)	Medium (~70)	Moderate (ORB + CNN)	85-90	Medium (Computer)	Medium	High
[6]	Deep Learning for Amputee Gesture Recognition	High (~100)	High (Deep Learning Model)	85-90	High (EMG Sensors)	Low	Low
[7]	Quantized CNN for Hardware-based Gesture Recognition	Low (~50)	Low (Quantized CNN)	90+	Low (Hardware-Based)	High	High
[8]	Virtual Finger-Point Behaviors (Mouse Cursor Tracking)	Low (~50)	Low (Cursor Tracking Algorithm)	85-90	Very Low (Software)	High	Very High
[9]	Hybrid-Metaheuristic and Deep Learning Model	High (~100)	Very High (Metaheuristics + CNN)	90-95	High (Computational)	Medium	Medium
[10]	Automated Gesture Recognition Using Time-Frequency Domain CNN	High (~150)	Very High (Time-Frequency + CNN)	90-95	High (Sensors)	Medium	Medium
Your Project	Virtual Mouse (Hand Gesture Control via Webcam)	Low (~30-50)	Low (Webcam + Simple Model)	90-95	Low (Webcam)	High	Very High


Latency Comparison (ms)
Latency (ms)
0
20
40
60
80
100
120
140
Paper 1
Paper 2
Paper 3
Paper 4
Paper 5
Paper 6
Paper 7
Paper 8
Paper 9
Paper 10
Your Project
Paper


Accuracy Comparison (%)
Accuracy (%)
0
20
40
60
80
100
Paper 1
Paper 2
Paper 3
Paper 4
Paper 5
Paper 6
Paper 7
Paper 8
Paper 9
Paper 10
Your Project
Paper


Computational Complexity Comparison (FLOPS)
Computational Complexity (FLOPS)
20000
17500
15000
12500
10000
7500
5000
2500
0
Paper 1
Paper 2
Paper 3
Paper 4
Paper 5
Paper 6
Paper 7
Paper 8
Paper 9
Paper 10
Your Project
Paper


Your Project Comparison (Radar Chart)
Accuracy
Computational Complexity
Latency
Feasibility
Generalization
Adaptability
Hardware Efficiency
Power Consumption

4.5 Discussions

The finger tracking and gesture recognition system demonstrated solid performance in enabling hands-free control of a computer through hand gestures. Key gestures, such as moving the mouse, left-clicking, and performing screenshots, were accurately recognized, offering an intuitive user experience. However, performance decreased in low-light environments, where the camera struggled with landmark detection, affecting accuracy and responsiveness. Complex gestures like two-finger scrolling and palm tilt scrolling showed higher error rates, as the system had difficulty tracking subtle hand movements. Multiple hand detection also posed challenges, leading to occasional misinterpretation when hands overlapped. Despite these limitations, the system effectively handled basic interactions with high accuracy. Future improvements could focus on refining gesture recognition algorithms, enhancing lighting conditions, and introducing machine learning for better performance in real-time applications. Integration of additional sensors or depth cameras may also improve accuracy. The system has strong potential for expansion into assistive technology and immersive computing experiences. Overall, the project successfully showcased the possibilities of gesture-based interaction, with room for further refinement.

5. Conclusion and Future Developments

A virtual mouse with hand gestures powered by AI presents an innovative application of computer vision and gesture recognition technologies to novel and friendly interfaces. Here, a project is described to the reader on how tools like OpenCV and MediaPipe can be applied so that hand movements are recognized and translated into actions done with traditional input devices. It ensures ease of access, and has a fallback option available for users interacting less with the conventional hardware, while also maximizing convenience and flexibility by minimizing reliance on tangible devices.

Methodologies Used:

- Hand Detection and Tracking (MediaPipe): Detects hands and extracts landmarks using machine learning models optimized for real-time performance.
- Gesture Recognition: Utilizes geometric analysis of detected landmarks (angles between points) to determine specific gestures.
- Action Mapping: Maps recognized gestures to specific cursor actions using PyAutoGUI, enabling interaction without physical contact.
- Smoothing Algorithms: Implements smoothing techniques to reduce jitter in cursor movements, enhancing user experience.

Further development may be towards accuracy and responsiveness in gesture recognition, especially in varied lighting conditions and different backgrounds. The inclusion of machine learning models for adaptive learning may even enable the system to understand specific unique gestures to enhance personalization and reliability. Additional gestures for voice command support and even more richly multi-finger gesture input may provide the richer experience for a fully multimodal environment that has boundaries yet to be touched in terms of what's possible hands-free both personally and in a professional arena.

6. Student Feedback (Student Experience in this Course Project)
Each team member can share your learning experience and others

Joel Joshua Richard: “This project was a great learning experience. I learned how to use computer vision and machine learning to build a gesture control system. Working with Python, OpenCV, and Mediapipe helped improve my coding skills. I also learned the importance of optimization for real-time systems, especially with gesture tracking."

Pranay Gorantla: ” I really enjoyed learning how gesture recognition works. The integration of hand tracking and gesture detection using a webcam was challenging but fun. I learned how to use libraries like Mediapipe and PyAutoGUI, and got better at Python. The project showed me how small changes in the code can improve performance."
Yashwanth R: “This was my first time applying machine learning to a real-world problem. It was tough to handle real-time data, but I learned a lot about TensorFlow and combining it with other libraries. I also learned how important it is to fine-tune models for specific tasks and optimize the system for real-time performance."

Dev Sandeep G.V: “I had little experience with computer vision before this, so it was exciting to work on hand tracking. I learned a lot about troubleshooting when things didn’t work as expected. Handling edge cases like partial hands or overlapping hands was tricky, but I got better at debugging.”

Praveen Chidambaram P : "I enjoyed working with the team and using hardware with Python. The real-time aspect made me think about how to optimize the system for smooth performance. It was a good lesson in teamwork and breaking down tasks into smaller parts. Seeing our project come together was rewarding."

7. References

1. Mohamed N, Mustafa MB, Jomhari N. A review of the hand gesture recognition system: Current progress and future directions. IEEE Access. 2021;9:157422-157436.

2. Gadekallu TR, Srivastava G, Liyanage M, Iyapparaja M, Chowdhary CL, Koppu S, et al. Hand gesture recognition based on a Harris hawks optimized convolution neural network. Comput Electr Eng. 2022;100:107836.

3. Zhang Z, Shen Q, Wang Y. Electromyographic hand gesture recognition using convolutional neural network with multi-attention. Biomed Signal Process Control. 2024;91:105935.

4. Wang S, Tang H, Chen F, Tan Q, Jiang Q. Integrated block-wise neural network with auto-learning search framework for finger gesture recognition using sEMG signals. Artif Intell Med. 2024;149:102777.

5. Damaneh MM, Mohanna F, Jafari P. Static hand gesture recognition in sign language based on convolutional neural network with feature extraction method using ORB descriptor and Gabor filter. Expert Syst Appl. 2023;211:118559.

6. Triwiyanto T, Pawana IPA, Caesarendra W. Deep learning approach to improve the recognition of hand gesture with multi-force variation using electromyography signal from amputees. Med Eng Phys. 2024;125:104131.

7. Jaiswal M, Sharma V, Sharma A, Saini S, Tomar R. Quantized CNN-based efficient hardware architecture for real-time hand gesture recognition. Microelectron J. 2024;106345.

8. Kirsh I. Virtual finger-point reading behaviors: A case study of mouse cursor movements on a website. Big Data Res. 2022;29:100328.

9. Mohyuddin H, Moosavi SKR, Zafar MH, Sanfilippo F. A comprehensive framework for hand gesture recognition using hybrid-metaheuristic algorithms and deep learning models. Array. 2023;19:100317.

10. Mohapatra AD, Aggarwal A, Tripathy RK. Automated recognition of hand gestures from multichannel EMG sensor data using time-frequency domain deep learning for IoT applications. IEEE Sens Lett. 2024.

11. Chen X, Li Y, Hu R, Zhang X, Chen X. Hand gesture recognition based on surface electromyography using convolutional neural network with transfer learning method. IEEE J Biomed Health Inform. 2020;25(4):1292-1304.

12. Malik J, Shimada S, Elhayek A, Ali SA, Theobalt C, Golyanik V, et al. Handvoxnet++: 3D hand shape and pose estimation using voxel-based neural networks. IEEE Trans Pattern Anal Mach Intell. 2021;44(12):8962-8974.

13. Tsai TH, Ho YC, Chi PT. Hardware architecture design for hand gesture recognition system on FPGA. IEEE Access. 2023;11:51767-51776.

14. Wang Z, Wang H, Yu H, Lu F. Interaction with gaze, gesture, and speech in a flexibly configurable augmented reality system. IEEE Trans Hum-Mach Syst. 2021;51(5):524-534.

15. Li J, Liu X, Wang Z, Zhang T, Qiu S, Zhao H, et al. Real-time hand gesture tracking for human–computer interface based on multi-sensor data fusion. IEEE Sens J. 2021;21(23):26642-26654.

16. Zanghieri M, Benatti S, Burrello A, Kartsch V, Conti F, Benini L. Robust real-time embedded EMG recognition framework using temporal convolutional networks on a multicore IoT processor. IEEE Trans Biomed Circuits Syst. 2019;14(2):244-256.

17. In S, Lin T, North C, Pfister H, Yang Y. This is the table I want! Interactive data transformation on desktop and in virtual reality. IEEE Trans Vis Comput Graph. 2023.

18. Terreran M, Barcellona L, Ghidoni S. A general skeleton-based action and gesture recognition framework for human–robot collaboration. Robotics Auton Syst. 2023;170:104523.

19. Rong Y, Gu G. Deep transfer learning-based adaptive gesture recognition of a soft e-skin patch with reduced training data and time. Sens Actuators A: Phys. 2023;363:114693.

20. Shanmugam S, Narayanan RS. An accurate estimation of hand gestures using optimal modified convolutional neural network. Expert Syst Appl. 2024;249:123351.

Video demonstration link:
https://drive.google.com/drive/folders/1OsT_zgTc181ptYglQDodUCcnGmhvIpfw?usp=sharing

Any other related information, you want to add.