# Hand Gesture Recognition for Controlling Computer's Mouse Cursor (JARVIS 4.0)

MAJOR PROJECT (CS-706) REPORT



Submitted by

Kundan Pandey Christafar Shibu

0902CS211029 0902CS211017

**B.Tech. (COMPUTER SCIENCE AND ENGINEERING)**

**(Batch 2021-25)**

Under the Guidance of

Kirtiraj Bhatele

Professor

**DECEMBER-2024**

Department Of Computer Science & Engineering

Rustamji Institute Of Technology, BSF Academy, Tekanpur

Department Of Computer Science & Information Technology

Rustamji Institute Of Technology, BSF Academy, Tekanpur



CERTIFICATE

This is to certify that this project report **JARVIS 4.0** was submitted by **Kundan Pandey and Christafar Shibu** who carried out the project work under my supervision. I approve this project for submission to the Bachelor of Engineering in the Department of Computer Science & Information Technology, Rustamji Institute of Technology, BSF Academy, Tekanpur.

|  |  |  |
| --- | --- | --- |
| Kirtiraj Bhatele  (Project Supervisor) | Prof. Vivek Gupta  (Major Project-I Incharge) | Dr. Jagdish Makhijani  (Head of Department) |

Dr. Chetan Pathak

Dean Academics

Principle

Dr. Kamlesh Gupta

ACKNOWLEDGEMENT

It gives me immense pleasure to express my deepest sense of gratitude and sincere thanks to my highly respected and esteemed guide (**Kirtiraj Bhatele, Computer Science and Engg. )**, Rustamji Institute of Technology, BSF Academy, Tekanpur, for their valuable guidance, encouragement and help for completing this work. Their useful suggestions for this whole work and cooperative behaviour are sincerely acknowledged.

I would like to sincerely thank Shri Ajeeth Kumar P., Chief Administrator, RJIT, Tekanpur for giving me this opportunity to undertake this project. I would also like to thank Dr. Kamlesh Kumar Gupta, Principal for their wholehearted support. I would also like to thank Dr. Rashmi Shah, Dean Academics for wholehearted support.

I also wish to express my gratitude to Dr. Jagdish Makhijani, Assistant Professor & HOD (Computer Science & Engineering) for his kind-hearted support.

I also wish to express my indebtedness to my parents as well as my family member whose blessings and support always helped me to face the challenges ahead.

Finally, I would like to express my sincere thanks to all my friends **(put the names of your friend to whom you want to pay gratitude)** and others who helped me directly or indirectly during this project work.

Place: RJIT, Tekanpur

# Abstract

Hand Gesture Recognition is a novel approach to Human-Computer Interaction (HCI). This project presents a Python-based application that leverages a webcam to recognize hand gestures and map them to mouse actions, enabling touch-free control of a computer cursor. By utilizing Mediapipe for hand tracking and PyAutoGUI for cursor control, this system supports various actions like moving, clicking, scrolling, and zooming. The project addresses accessibility challenges and promotes intuitive, contactless computing. However, it currently operates under specific lighting and environmental conditions. Future work can include multi-hand recognition and adaptability to dynamic environments.

# Table of Contents

1. Introduction  
2. Analysis  
3. Design  
4. Implementation  
5. Conclusion & Future Work  
6. References

# Chapter I: Introduction

**Background and Motivation**

Human-Computer Interaction (HCI) has evolved significantly since its inception, with the primary goal of improving how users interact with computers. Traditional interfaces, such as keyboards and mice, have served as the cornerstone of this interaction for decades. However, the limitations of these interfaces have become apparent in modern computing, especially with the rise of smart devices, virtual environments, and accessibility needs.

Gesture-based systems emerged as a response to the demand for more intuitive and natural ways of interaction. These systems leverage the human body as an input device, allowing users to control digital interfaces through movements and gestures. This approach not only enhances user experience but also reduces dependency on physical devices, making computing more inclusive.

In recent years, gesture recognition has gained popularity due to advancements in computer vision, machine learning, and hardware capabilities. From gaming consoles like the Microsoft Kinect to smart TVs and mobile devices, gesture recognition is now an integral part of many industries. Its relevance extends to areas like accessibility for differently-abled individuals, contactless control in medical environments, and immersive virtual reality experiences.

The motivation behind this project is to develop a gesture-based system that bridges the gap between humans and machines, creating a seamless and efficient interaction method. The project focuses on controlling a computer's mouse cursor through hand gestures, providing a touch-free alternative to traditional devices. This solution is particularly significant in scenarios where hygiene and accessibility are critical, such as hospitals, laboratories, or public workstations.

**Problem Statement**

Traditional input devices like keyboards and mice are limited by their reliance on physical contact, which can pose challenges in certain situations. For instance, in hygiene-critical environments such as hospitals, laboratories, or public kiosks, physical devices can become vectors for germs and pathogens. Similarly, individuals with mobility impairments may find it challenging or impossible to use conventional input devices.

With the increasing reliance on digital interfaces in every aspect of life, there is a growing need for innovative interaction methods that overcome these limitations. Gesture recognition offers a promising solution by enabling touch-free interaction. However, existing gesture recognition systems often suffer from issues like:

* Low accuracy in detecting gestures under diverse conditions.
* Inconsistent performance in real-time applications.
* Limited adaptability to different user preferences and environments.

This project addresses these challenges by developing a real-time hand gesture recognition system for controlling a computer's mouse cursor. The system is designed to operate using a standard webcam, making it accessible and cost-effective. By leveraging modern libraries like Mediapipe for gesture detection and PyAutoGUI for mouse control, this project aims to create a robust, intuitive, and efficient solution.

# Chapter II: Analysis

**Problem Analysis**

Detecting and mapping hand gestures reliably in real-time is a complex task that involves several technical and practical challenges. These challenges can be broadly categorized into the following areas:

**1. Real-Time Gesture Detection**

* **Accuracy and Speed**: Detecting gestures with high accuracy while maintaining low latency is critical for ensuring a seamless user experience. Even minor delays or inaccuracies can lead to frustration and reduce system usability.
* **Landmark Detection**: Reliable detection of hand landmarks (e.g., fingertips, knuckles) is essential for distinguishing between different gestures. Variations in hand size, orientation, or position relative to the webcam can complicate this process.
* **Lighting and Background Noise**: Changes in ambient lighting, shadows, or complex backgrounds can adversely affect the system's ability to detect hands accurately.

**2. Gesture Differentiation**

* **Similar Gestures**: Some gestures may have subtle differences that are difficult to distinguish, especially if the user performs them inconsistently or quickly.
* **Dynamic Gestures**: Recognizing gestures that involve movement (e.g., scrolling or dragging) requires tracking hand motion across consecutive video frames, which increases computational complexity.
* **User Variability**: Differences in hand shapes, sizes, and movement styles among users can make gesture recognition less consistent.

**3. Environmental Factors**

* **Lighting Conditions**: Poor or uneven lighting can lead to incomplete or noisy data, making it harder for the system to detect landmarks.
* **Camera Quality**: Variations in webcam resolution and frame rate can impact the accuracy and speed of gesture detection.
* **Occlusion**: Partial occlusion of the hand (e.g., by objects or poor positioning) can disrupt the detection process.

**4. System Integration**

* **Mapping to Mouse Actions**: Translating recognized gestures into precise and natural mouse actions is a key challenge. The mapping must feel intuitive to users and provide accurate control of the cursor or other functions.
* **Software Compatibility**: Ensuring that the system works seamlessly across different operating systems and hardware configurations requires careful design and testing.

**Requirements**

To address the above challenges, the project requires specific hardware and software components. Here is a detailed breakdown:

**Hardware**

1. **Webcam**:
   * A standard high-definition (HD) webcam with a minimum resolution of 720p is recommended. Higher resolutions provide more accurate detection of hand landmarks.
   * Frame rate: A webcam capable of capturing at least 30 frames per second (FPS) ensures smoother detection and responsiveness.
2. **Computer**:
   * A computer with a modern processor (e.g., Intel i5 or equivalent) to handle real-time video processing.
   * At least 4GB of RAM is required to support the simultaneous execution of Mediapipe and PyAutoGUI.
   * A stable operating system such as Windows, macOS, or Linux.

**Software**

1. **Python (3.x)**:
   * The programming language used for implementing the application. Python's extensive library ecosystem and ease of use make it ideal for rapidly developing computer vision projects.
2. **OpenCV**:
   * An open-source computer vision library used for video capture and preprocessing. OpenCV provides functions for frame extraction, colour space conversion, and basic image processing tasks.
3. **Mediapipe**:
   * A machine learning framework by Google that provides pre-trained real-time hand detection and tracking models. Mediapipe’s hand module detects 21 key landmarks on the hand and assigns 3D coordinates to each point.
4. **PyAutoGUI**:
   * A Python library for programmatically controlling the mouse and keyboard. PyAutoGUI is used to map gestures detected by Mediapipe to corresponding mouse actions, such as cursor movement, clicks, scrolling, and dragging.

# Chapter III: Design

**System Architecture**

The system architecture for the hand gesture recognition project is designed to enable seamless interaction between the user and the computer. It is divided into three key layers: **Input Layer**, **Processing Layer**, and **Action Layer**. Each layer is responsible for specific functionalities that collectively contribute to the system’s performance and usability.

**1. Input Layer: Webcam**

The input layer is the foundation of the system, responsible for capturing the raw video stream containing the user’s hand gestures. This layer provides the primary data source for gesture recognition and cursor control.

* **Role of the Webcam**:
  + The webcam continuously captures video frames of the user’s hand movements.
  + A standard HD webcam is sufficient for most scenarios, but higher resolution cameras can enhance detection accuracy.
  + The video stream is processed in real-time to detect and track hand landmarks.
* **Challenges at the Input Layer**:
  + Environmental factors such as lighting, shadows, and background clutter can affect the clarity of captured frames.
  + Variations in user hand positioning relative to the webcam may introduce challenges in consistent gesture detection.
* **Optimization Considerations**:
  + Position the webcam to ensure clear visibility of the hand within its field of view.
  + Adjust lighting to minimize shadows and enhance contrast for better detection.

**2. Processing Layer: Mediapipe for Hand Tracking**

The processing layer is the core of the system, where the captured video frames are analyzed to detect and recognize hand gestures. This layer relies on the **Mediapipe** library, which provides advanced machine learning-based models for real-time hand tracking.

* **How Mediapipe Works**:
  + Mediapipe’s hand detection module identifies the user’s hand in the video frame and extracts **21 key landmarks** on the hand, including fingertips, knuckles, and the wrist.
  + Each landmark is represented by 3D coordinates (x, y, z), where:
    - x and y indicate the position of the landmark in the video frame.
    - z provides depth information, helping to determine gestures like pinching or spreading fingers.
  + The library is optimized for low-latency processing, making it suitable for real-time applications.
* **Key Tasks in the Processing Layer**:
  + **Preprocessing**: Convert video frames to a suitable format (e.g., RGB) for Mediapipe processing.
  + **Hand Detection**: Identify the presence and position of the hand in each frame.
  + **Landmark Extraction**: Track the movement of key landmarks across consecutive frames to interpret gestures.
* **Features of Mediapipe**:
  + Robust detection under varying conditions, including different hand sizes and orientations.
  + Efficient use of computational resources, enabling real-time performance even on standard hardware.
* **Challenges and Considerations**:
  + Managing occlusion, where parts of the hand may be obscured by other objects or fingers.
  + Ensuring stability and consistency of landmark tracking during rapid hand movements.

**3. Action Layer: PyAutoGUI**

The action layer translates recognized gestures into corresponding mouse actions, providing the user with a seamless interaction experience. This layer uses the **PyAutoGUI** library to control the mouse cursor programmatically.

* **Role of PyAutoGUI**:
  + PyAutoGUI receives gesture classifications (e.g., "move cursor," "left click") from the processing layer.
  + It maps these gestures to mouse functions, enabling actions like moving the cursor, clicking, scrolling, and dragging.
  + The library interacts directly with the operating system, making it compatible with most platforms.
* **Examples of Gesture-to-Action Mappings**:
  + **Open Hand**: All fingers extended → Move the cursor in real-time based on hand position.
  + **Fist**: All fingers folded → Drag and drop objects by holding the left mouse button.
  + **Index Finger Raised**: Single finger extended → Perform a left-click.
  + **Pinching Motion**: Thumb and index finger move together → Zoom in.
* **Challenges at the Action Layer**:
  + Ensuring smooth and responsive cursor movement without jitter.
  + Preventing accidental clicks or unintended actions due to gesture misclassification.
* **Optimization Techniques**:
  + Introduce thresholds or filters to ignore minor hand movements or unintentional gestures.
  + Calibrate gesture sensitivity to match user preferences, providing a balance between responsiveness and accuracy.

# Chapter IV: Implementation

**Steps to Execute**

The following steps outline the process to set up and execute the hand gesture recognition system:

**1. Install Libraries Using Pip**

To run the project, you need to install the required Python libraries. Use the commands below to install the dependencies.

1. Open a terminal or command prompt.
2. Run the following command to install the required libraries individually:

bash

Copy code

pip install opencv-python mediapipe pyautogui

1. Alternatively, if the project includes a requirements.txt file, you can install all dependencies in one step:

bash

Copy code

pip install -r requirements.txt

This ensures that the correct versions of each library are installed, avoiding compatibility issues.

**Dependencies Explained**:

* **OpenCV**: For video capture and basic image processing.
* **Mediapipe**: For real-time hand detection and landmark extraction.
* **PyAutoGUI**: For mapping gestures to mouse actions.

**2. Prepare Project Structure**

Organize the project files in a structured manner to ensure smooth execution and maintenance. Below is an example of the recommended directory structure:

Hand Gesture

├── app.py # Main script for capturing video and recognizing gestures

├── controller.py # Handles mouse actions (e.g., clicks, scrolling)

├── requirements.txt # List of required Python libraries

**3. Run app.py**

The app.py file is the main entry point for the project. Follow these steps to execute it:

1. Open a terminal or command prompt and navigate to the project directory:

>> cd HandGestureRecognition

1. Run the application:

>> python app.py

1. Upon execution:
   * The webcam feed will open, and the system will start detecting hand gestures.
   * Make sure your hand is visible within the webcam’s field of view.
   * Perform gestures to control the mouse cursor as described in the gesture mapping section.
2. press the ESC key or close the webcam window to stop the application.

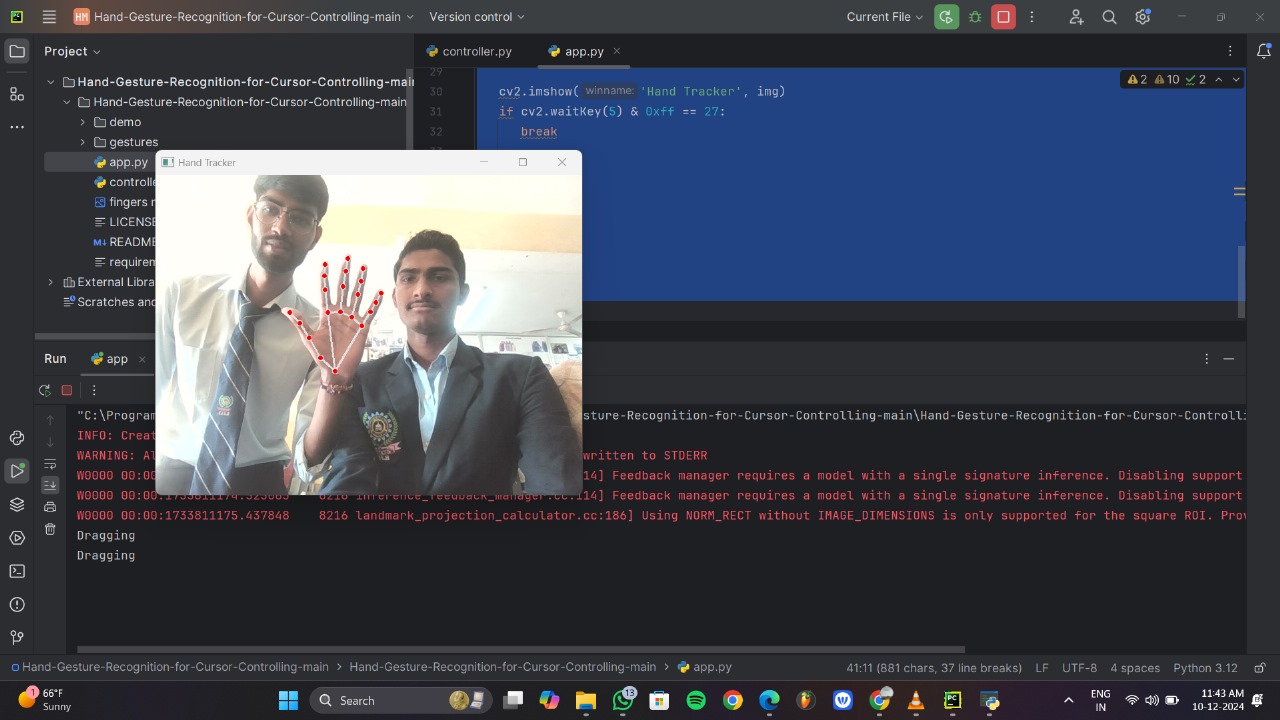
# Chapter V: Conclusion & Future Work

**Summary of Accomplishments**

This project successfully demonstrates the feasibility and effectiveness of using hand gestures to control a computer's mouse cursor. The primary accomplishments of the project are summarized as follows:

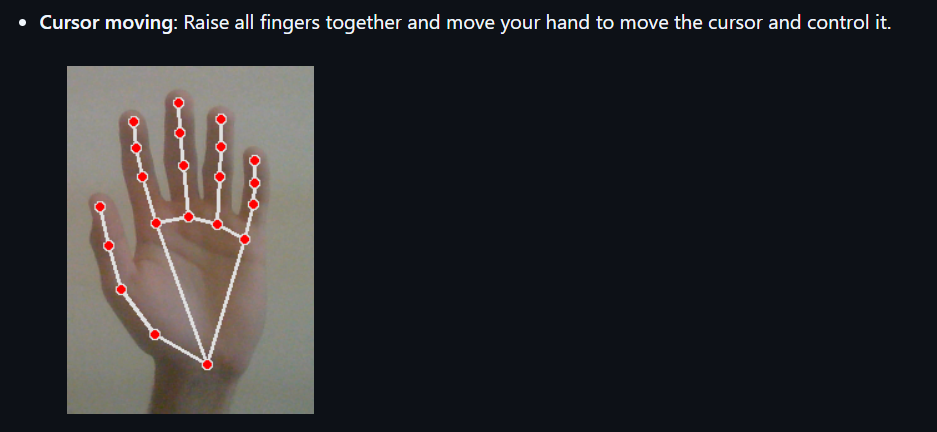
1. **Real-Time Hand Gesture Recognition**:  
   The system leverages Mediapipe’s robust hand tracking module to achieve real-time detection and recognition of hand gestures. It accurately identifies and tracks 21 hand landmarks with minimal latency, ensuring a smooth and responsive interaction experience.
2. **Mapping Gestures to Mouse Functions**:  
   Predefined gestures are effectively mapped to mouse actions using PyAutoGUI. Users can perform cursor movements, clicks, scrolling, and zooming by executing simple and intuitive gestures. This mapping provides a natural and user-friendly interface.
3. **Touch-Free Interaction**:  
   The project provides a contactless alternative to traditional input devices, making it ideal for hygiene-critical environments such as hospitals, laboratories, and shared public workstations.
4. **Cost-Effective Solution**:  
   By utilizing readily available hardware (a standard webcam) and open-source libraries, the project demonstrates a low-cost implementation without compromising on functionality.
5. **Flexibility and Adaptability**:  
   The modular architecture allows for easy modification and extension of the system. New gestures can be added or existing gestures redefined without major changes to the codebase.
6. **Accessibility Improvements**:  
   The project has the potential to enhance accessibility for individuals with mobility impairments, offering an alternative to conventional input devices like keyboards and mice.
7. **Proof of Concept for Future Development**:  
   This project lays a strong foundation for further advancements in gesture recognition and human-computer interaction, providing a working model that can be scaled and enhanced.

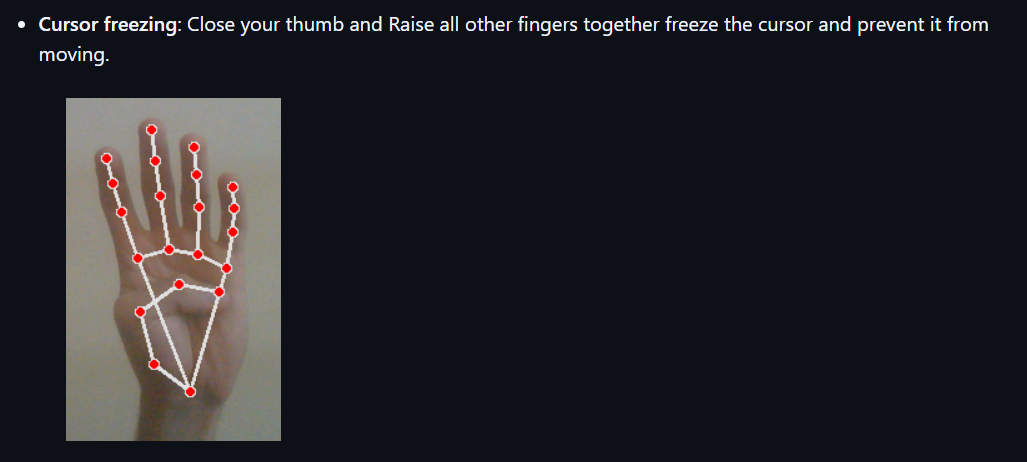
# TESTING:

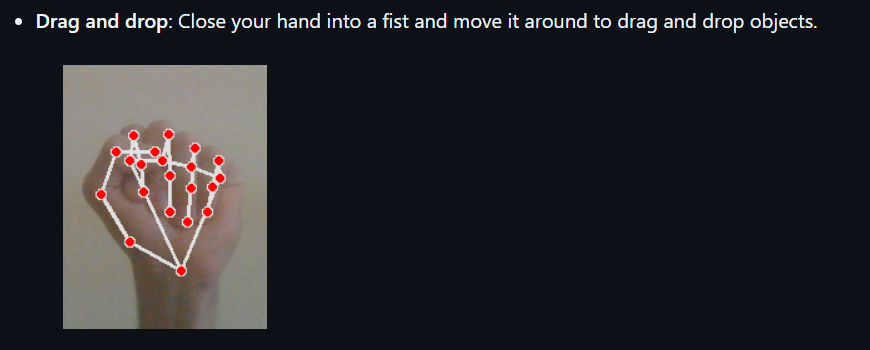


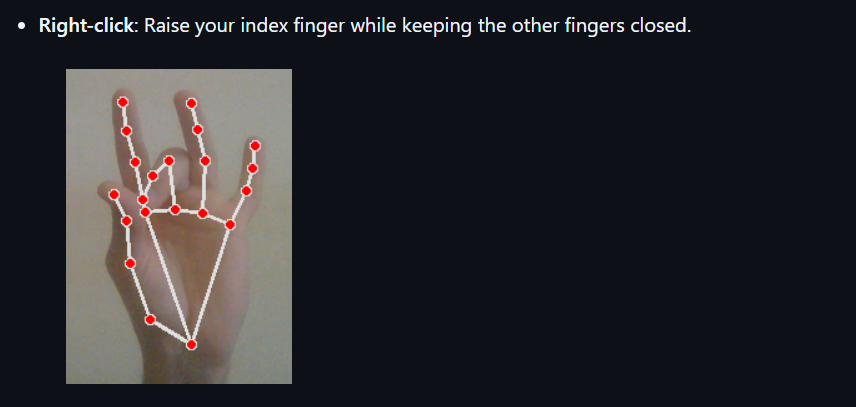
# 

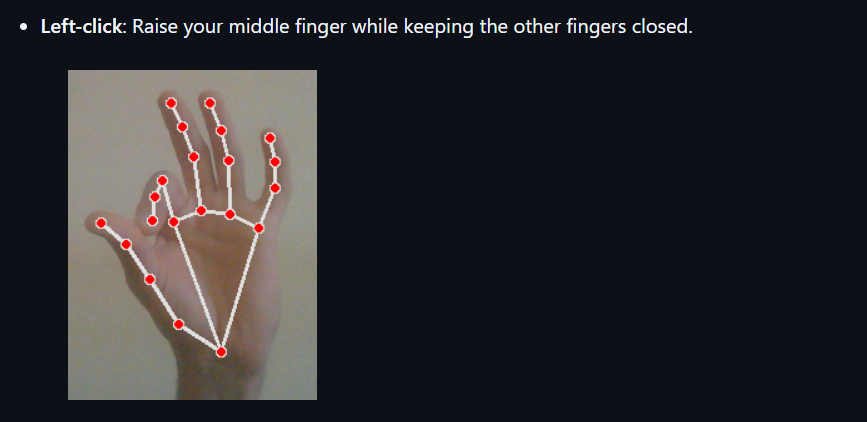
# Appendices

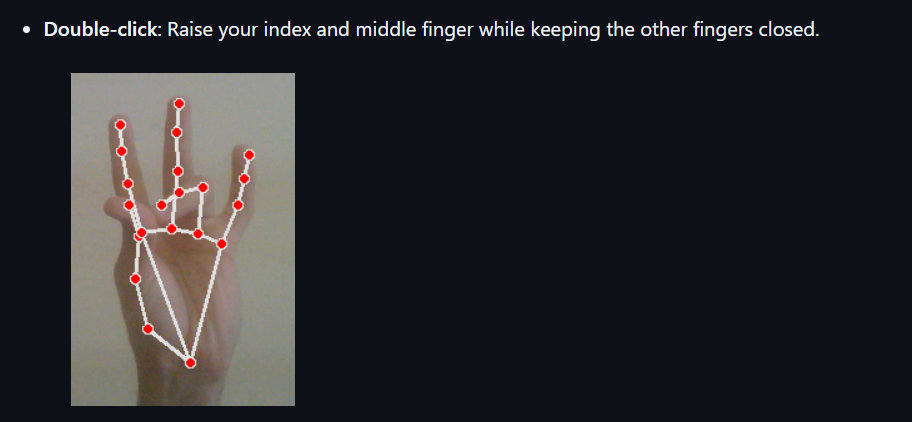


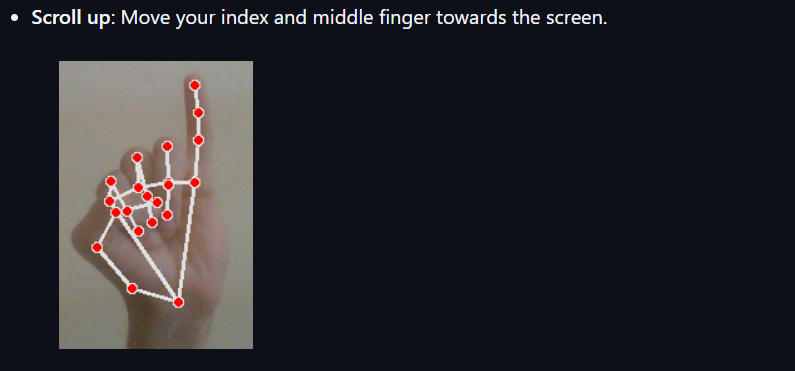


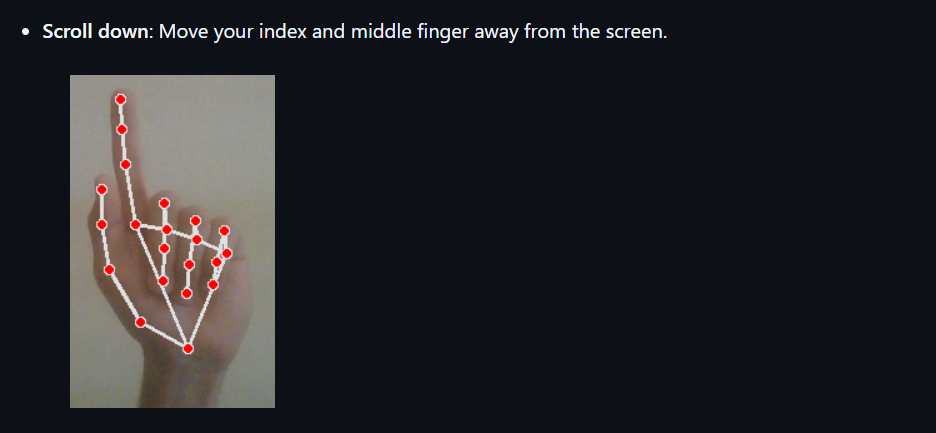


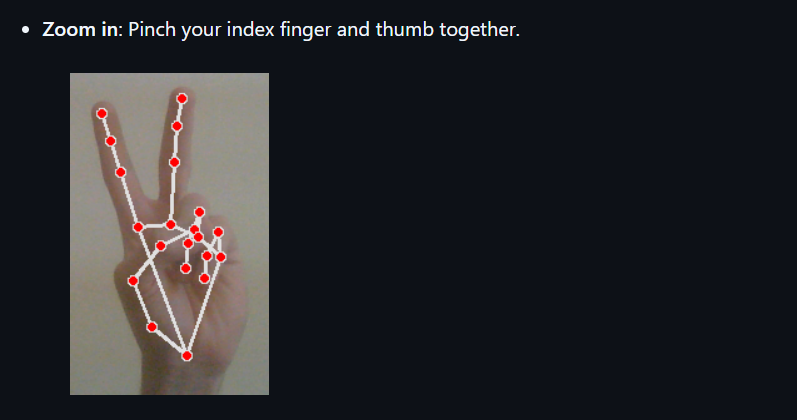


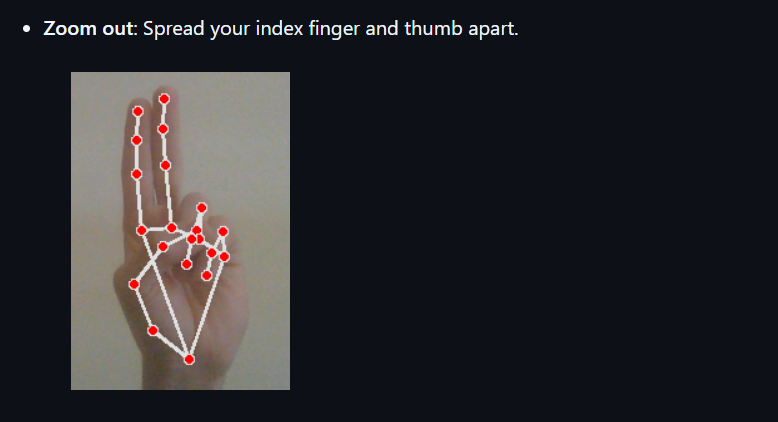












# References

1 Mediapipe Documentation:  
 <https://mediapipe.dev/>

2 PyAutoGUI Documentation:  
 <https://pyautogui.readthedocs.io/>

3 OpenCV Tutorials:  
 <https://opencv.org/>

# Appendices

**Appendix A: Project Directory Structure**

The project is organized as follows:

HandGestureRecognition/

├── app.py # Main script for capturing video and recognizing gestures

├── controller.py # Handles mouse actions (e.g., clicks, scrolling)

├── requirements.txt # List of required Python libraries

**Appendix B: Sample Code Snippets**

1. **Gesture Recognition Logic (app.py)**

**>>**

import mediapipe as mp

import cv2

mp\_hands = mp.solutions.hands

hands = mp\_hands.Hands(max\_num\_hands=1, min\_detection\_confidence=0.7)

cap = cv2.VideoCapture(0)

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

frame = cv2.flip(frame, 1) # Mirror the frame

rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

results = hands.process(rgb\_frame)

if results.multi\_hand\_landmarks:

for hand\_landmarks in results.multi\_hand\_landmarks:

# Process gesture recognition here

print(hand\_landmarks)

cv2.imshow("Hand Gesture Recognition", frame)

if cv2.waitKey(1) & 0xFF == 27: # Exit on ESC

break

cap.release()

cv2.destroyAllWindows()

This appendix section provides supplementary information, making the project more comprehensive and user-friendly for implementation and understanding.