School of Computer Science Engineering & Information Systems (SCORE)

M. Tech Software Engineering

Winter Semester 2024-25

PROJECT REPORT

On

Hand Gesture Controlled Virtual Mouse using Fuzzy Logic

Submitted in fulfilment for the JComponent of

SWE1011 - Soft Computing

Under the guidance of

Dr. HEMALATHA. S

Submitted by,

GANESH.P– 22MIS0525
SIDDHARTH NAIR-22MIS0097
PRIYAN J-21MIS0381
MOHAMMED BILAL K – 21MIS0082

Table of Contents

1. Abstract

This project presents a real-time hand gesture recognition system for virtual mouse control. The system utilizes computer vision techniques and fuzzy logic to accurately detect and interpret hand gestures, such as mouse movement, clicks, and screenshot. By leveraging the power of fuzzy logic, the system can handle the inherent variability and uncertainty in hand gestures, resulting in robust and reliable performance. The system has potential applications in various fields, including gaming, accessibility, and human-computer interaction. The project provides a novel approach to controlling mouse functions through hand gesture recognition, utilizing a fuzzy logic-based system. By leveraging computer vision techniques, the system accurately analyzes finger angles and configurations, enabling intuitive and touchless interaction with a computer. A fuzzy inference system is employed to interpret these hand gestures, translating them into precise mouse movements, clicks, and screenshots. The proposed method offers several advantages, including adaptability to varying hand sizes and gestures, robustness to noise and occlusion, and real-time performance. Experimental results demonstrate the system's effectiveness in achieving accurate and responsive control, significantly enhancing human-computer interaction.

Keywords - Hand Gesture Recognition, Fuzzy Logic, Mouse Control, Soft Computing, Human-Computer Interaction (HCI), Membership Functions, Fuzzy Inference System (FIS), Computer Vision, Touchless Interaction, Gesture-Based Control

Chapter 1: Introduction

1.1 **General**

Hand Gesture Recognition

Hand gesture recognition is a field of computer vision that focuses on interpreting human gestures using cameras or other sensors. It involves capturing images or video sequences of hand movements, processing them to extract relevant features, and then classifying these features into specific gestures. This technology has gained significant attention in recent years due to its potential applications in various domains.

Importance of Gesture Recognition in Accessibility and Human-Computer Interaction

- **Accessibility:** Gesture recognition can provide a more intuitive and accessible way for people with disabilities to interact with computers. For instance, individuals with motor impairments can control devices using hand gestures, eliminating the need for physical input devices like keyboards and mice.
- **Human-Computer Interaction:** Gesture recognition enhances human-computer interaction by enabling natural and intuitive communication. It allows users to control devices and software applications using gestures, making the interaction more engaging and efficient.

Fuzzy Logic and Its Role in Soft Computing

Fuzzy logic is a form of multi-valued logic that deals with reasoning with imprecise information. Unlike traditional binary logic, where variables can only take on the values of true or false (1 or 0), fuzzy logic allows for degrees of truth. This makes it suitable for dealing with real-world problems that involve uncertainty and ambiguity.

In the context of soft computing, fuzzy logic plays a crucial role in modelling human reasoning and decision-making processes. It provides a framework for handling imprecise and vague information, which is often encountered in real-world scenarios. By employing fuzzy logic, systems can make more intelligent and flexible decisions, especially in situations where precise information is not available.

By combining fuzzy logic with computer vision techniques, hand gesture recognition systems can effectively handle the inherent uncertainties and variations in human gestures, leading to more accurate and robust recognition.

1.2 **Objective and Scope of the Project**

Objective:

The primary objective of this project is to develop a robust and efficient hand gesture recognition system that can be used to control a computer mouse virtually. This system will leverage the power of fuzzy logic to handle the inherent uncertainties and variations in human gestures, ensuring accurate and reliable performance.

Scope:

The scope of this project encompasses the following:

- **Real-time Hand Gesture Recognition:** Accurate and timely detection of predefined hand gestures, including mouse movement, left-click, right-click, and screenshots, and others.
- **Fuzzy Logic Implementation:** Design and implementation of a fuzzy logic system to handle the imprecise and ambiguous nature of hand gestures.
- **Virtual Mouse Control:** Integration of the gesture recognition system with a virtual mouse to enable control of the computer cursor and perform mouse clicks.
- **User Interface:** Development of a user-friendly interface to display recognized gestures and system status.
- **Performance Evaluation:** Assessment of the system's performance in terms of accuracy, speed, and robustness.
- **Accessibility:** Exploration of potential applications in accessibility, such as enabling individuals with disabilities to interact with computers more efficiently.
- **Gaming:** Investigation of the use of gesture-based controls in gaming for enhanced immersion and gameplay experiences.
- **Hands-free Computing:** Development of a hands-free computing environment where users can interact with computers without physical input devices.

Chapter 2: Literature Review

2.1 **General**

Review of Existing Gesture Recognition Technologies

Hand gesture recognition has been an active area of research for several decades. Various techniques have been proposed to detect and interpret hand gestures, each with its own advantages and limitations. Some of the most common approaches include:

Computer Vision-based Methods:

- **Traditional Image Processing:** Techniques like edge detection, feature extraction, and template matching have been used to analyse hand images and identify gestures. However, these methods are often sensitive to noise and variations in lighting conditions.
- **Machine Learning:** Machine learning algorithms, such as Support Vector Machines (SVM), Neural Networks, and Hidden Markov Models (HMM), have been employed to classify hand gestures based on features extracted from image sequences.
- **Deep Learning:** Deep learning, particularly Convolutional Neural Networks (CNNs), has revolutionized the field of computer vision, enabling highly accurate and robust hand gesture recognition. CNNs can automatically learn complex features from large amounts of training data.

Sensor-based Methods:

- **Wearable Sensors:** Devices like gloves or wristbands equipped with sensors (e.g., accelerometer, gyroscope) can capture hand movements and translate them into digital signals.
- **Depth Sensors:** Depth sensors, such as Kinect or Intel RealSense, can capture 3D depth information, allowing for more accurate and robust gesture recognition.

Fuzzy Logic Applications in Human-Computer Interaction

Fuzzy logic has been successfully applied in various aspects of human-computer interaction, including:

- **Natural Language Processing:** Fuzzy logic can be used to model the inherent ambiguity and imprecision in natural language, enabling more accurate and context-aware language processing.
- **Intelligent User Interfaces:** Fuzzy logic can be employed to create intelligent user interfaces that can adapt to user preferences and behaviour, providing a more personalized and intuitive experience.
- **Virtual Reality and Augmented Reality:** Fuzzy logic can be used to model human perception and cognition, leading to more realistic and immersive virtual and augmented reality experiences.

In the context of hand gesture recognition, fuzzy logic can be used to:

- **Handle Uncertainty and Imprecision:** Fuzzy logic can effectively handle the inherent uncertainty and imprecision associated with human gestures, such as variations in hand shape, size, and movement speed.
- **Model Human Perception:** Fuzzy logic can be used to model human perception of hand gestures, allowing for more accurate and robust recognition.
- **Improve System Robustness:** By incorporating fuzzy logic, hand gesture recognition systems can become more tolerant to noise, variations in lighting conditions, and other environmental factors.

2.2 Studies on Virtual Mouse Systems

Key Research Papers and Projects

A significant amount of research has been conducted in the field of virtual mouse systems, particularly focusing on gesture-based interfaces. Some notable studies include:

Vision-based Gesture Recognition:

- **Real-time Hand Gesture Recognition for Human-Computer Interaction:** This research explores the use of computer vision techniques to recognize hand gestures in real-time. However, it often requires significant computational resources and can be sensitive to lighting conditions and background clutter.
- **Hand Gesture Recognition Using Deep Learning:** Deep learning-based approaches, such as Convolutional Neural Networks (CNNs), have shown promising results in hand gesture recognition. However, they require large datasets for training and can be computationally expensive.

Sensor-based Gesture Recognition:

- **Gesture-Based Interaction Using Wearable Sensors:** This research investigates the use of wearable sensors, like inertial measurement units (IMUs), to capture hand movements and translate them into virtual mouse actions. While these systems can be accurate and robust, they require users to wear additional devices, which may be inconvenient.

2.3 **Gaps and Limitations Addressed by Our Project**

While existing research has made significant progress in virtual mouse systems, there are still several gaps and limitations that our project aims to address:

- **Robustness to Environmental Factors:** Many existing systems are sensitive to variations in lighting conditions, background clutter, and camera angle. Our project aims to improve the robustness of the system by using fuzzy logic to handle uncertainties and variations in hand gestures.
- **Real-time Performance:** Real-time performance is crucial for a seamless user experience. Our project focuses on optimizing the system's processing pipeline to achieve real-time gesture recognition.
- **User-friendliness:** The system should be easy to use and intuitive. We aim to design a user-friendly interface that minimizes user effort and maximizes efficiency.
- **Accessibility:** Our project aims to make virtual mouse systems more accessible to people with disabilities by providing a flexible and customizable interface.
- **Versatility:** The system should be versatile and adaptable to different user preferences and use cases. We will explore different gesture sets and customization options to cater to diverse user needs.

By addressing these gaps and limitations, our project aims to contribute to the advancement of virtual mouse systems, making them more reliable, efficient, and accessible.

Chapter 3: Methodology

3.1 System Overview & Architecture:

The proposed system operates in a real-time manner, continuously capturing video frames from a webcam or built-in camera. Each frame is processed through the following stages:

1. Camera Input:

- A webcam or built-in camera is used to capture video frames of the user's hand gestures.

2. Hand Detection and Landmark Extraction:

- **MediaPipe Hands:** The MediaPipe Hands solution, a machine learning pipeline, is employed to detect and track hands in each frame.
- **Landmark Extraction:** Key landmarks on the hand, such as the fingertips and joints, are extracted from the detected hand.

3. Feature Extraction:

- **Distance and Angle Calculations:** Relevant features, such as the distance between the index fingertip and thumb tip, and the angles between fingers, are calculated based on the extracted landmarks. These features provide insights into the shape and orientation of the hand

4. Fuzzy Logic-based Gesture Classification:

- **Input Variables:** The extracted features are fed as input to the fuzzy logic system.
- **Membership Functions:** Fuzzy membership functions are defined for each input variable to represent the degree of membership to different linguistic terms (e.g., "small," "medium," "large" for distance).
- **Fuzzy Rules:** Fuzzy rules are defined to map input values to output gestures. For example, a rule might state: "IF distance is small AND index finger angle is large THEN gesture is left click."
- **Fuzzy Inference:** The fuzzy inference engine processes the input values and the fuzzy rules to produce a fuzzy output.
- **Defuzzification:** The fuzzy output is defuzzified to obtain a crisp output value, which represents the recognized gesture.

5. Action Execution:

- **Virtual Mouse Control:** Based on the recognized gesture, appropriate actions are triggered, such as moving the mouse cursor, clicking, screenshot, or performing other predefined commands.
- **User Interface:** A user interface can be implemented to provide visual feedback on the recognized gestures and system status.

3.2 Tools and Libraries

Several software tools and libraries were utilized to achieve gesture recognition and control of the virtual mouse. Each tool and library served a specific purpose in implementing the system, as described below:

MediaPipe

Purpose: MediaPipe is a cross-platform framework for building multimodal machine learning pipelines.

Usage in the Project:

- The **Hands** module from MediaPipe was used to detect and track hand landmarks in real-time from the video stream.
- It provided precise locations of 21 hand landmarks, including fingertips, which were crucial for calculating distances and angles required for gesture recognition.

scikit-fuzzy

Purpose: scikit-fuzzy is a Python library for fuzzy logic computations, including membership function definitions and fuzzy inference systems.

Usage in the Project:

- Used to implement the fuzzy logic-based gesture recognition system.
- Membership functions were defined for inputs such as **distance** and **finger angles**, as well as for the output gestures (e.g., move, click, minimize).
- Fuzzy rules were implemented to map inputs to gestures, enabling a more intuitive and human-like interaction system.

OpenCV

Purpose: OpenCV (Open-Source Computer Vision Library) is a popular library for real-time computer vision tasks.
Usage in the Project:

- Used to process video input from the webcam.
- Frames were converted to RGB format for compatibility with MediaPipe.
- Hand landmarks were drawn on the frames to provide visual feedback on gesture recognition.

PyAutoGUI

Purpose: PyAutoGUI is a library that allows programmatic control of the mouse and keyboard.
Usage in the Project:

- Used to simulate mouse movements and clicks based on recognized gestures.
- Implemented actions like left-click, right-click, taking screenshots, and minimizing windows by mapping specific gestures to these actions.

pynput

Purpose: pynput is a library for controlling and monitoring input devices such as keyboards and mice.
Usage in the Project:

- Used as an alternative to PyAutoGUI for certain mouse actions, such as clicking events.

psutil

Purpose: psutil is a library for system and process monitoring.
Usage in the Project:

- Used to monitor CPU and memory usage during the system's operation, providing performance metrics to evaluate the efficiency of the program.

win32gui

Purpose: win32gui is a library that provides functions for interacting with the Windows GUI.
Usage in the Project:

- Used to minimize the active window upon recognizing the corresponding gesture.

NumPy

Purpose: NumPy is a library for numerical computations in Python.
Usage in the Project:

- Used for creating arrays and performing calculations on distances, angles, and membership functions.
- Integral for fuzzy logic operations in the scikit-fuzzy library.

time

Purpose: A built-in Python library for handling time-related tasks.
Usage in the Project:

- Used for calculating performance metrics like FPS (Frames Per Second) and total runtime.
- Essential for measuring the efficiency of gesture recognition and mouse control.

csv

Purpose: A built-in Python library for working with CSV (Comma Separated Values) files.
Usage in the Project:

- If implemented, this library would store performance metrics, session logs, or gesture recognition results in a structured format.
- Helps maintain a history of session statistics for further analysis.

By integrating these tools and libraries, the system achieved real-time performance and effective gesture recognition for controlling mouse actions. This combination highlights the versatility and power of Python libraries in soft computing applications.

3.3 **Fuzzy Logic Implementation**

3.3.1 **Input Variables**

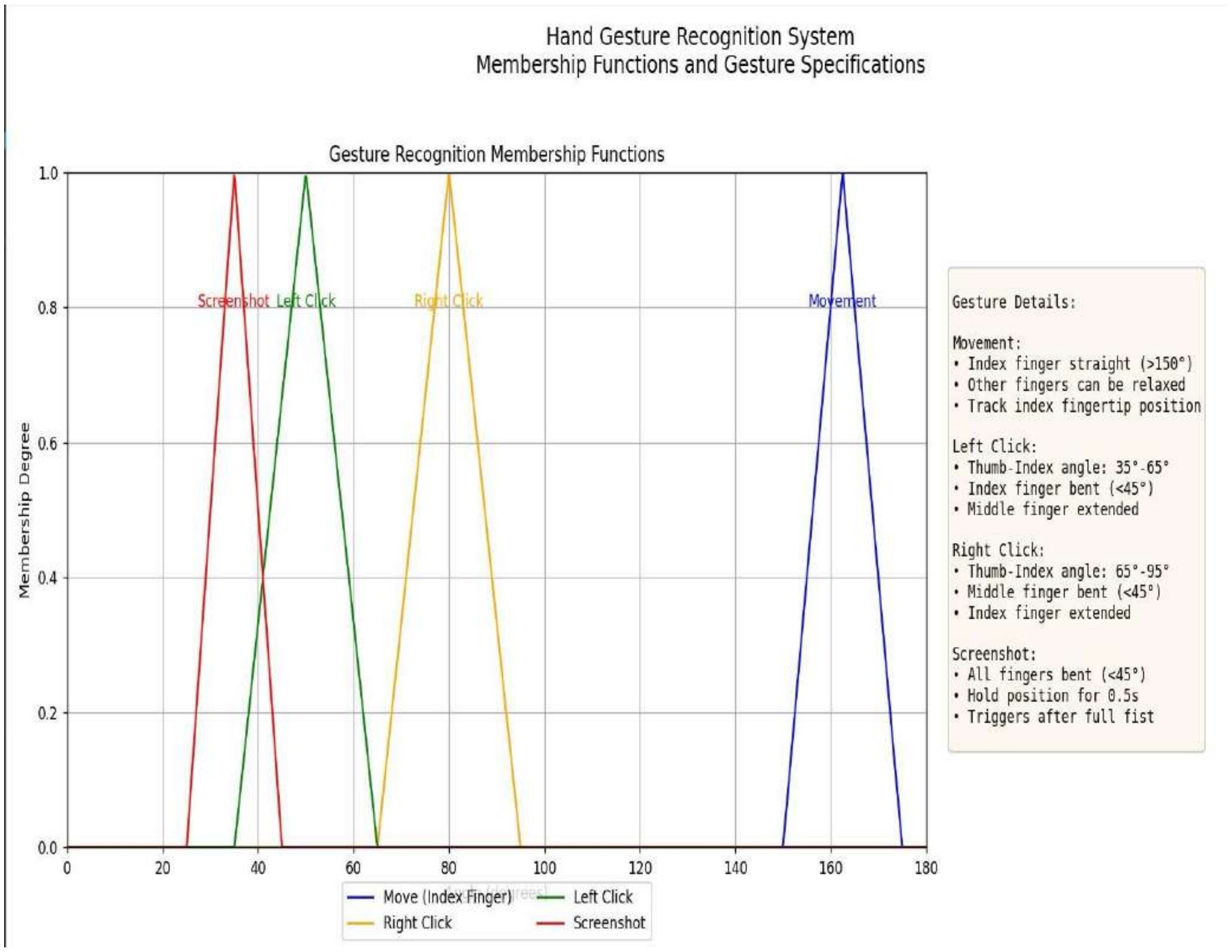
In the provided code, the following key features are extracted from the hand landmarks to serve as input variables for the fuzzy logic system:

1. **Distance between Thumb and Index Finger:** This distance is used to differentiate between gestures like clicking and others.
2. **Finger Angles:** The angles between finger segments are used to identify specific gestures, such as a closed or bended fingers for screenshot.

3.3.2 **Membership Function Design**

Membership functions define the degree to which a given input value belongs to a fuzzy set. In this project, triangular membership functions are used to represent the linguistic terms "small," "medium," and "large" for distance and angle values.

Distance Membership Functions:



Gesture Details:

Movement:
• Index finger straight (>150°)
• Other fingers can be relaxed
• Track index fingertip position

Left Click:
• Thumb-Index angle: 35°-65°
• Index finger bent (<45°)
• Middle finger extended

Right Click:
• Thumb-Index angle: 65°-95°
• Middle finger bent (<45°)
• Index finger extended

Screenshot:
• All fingers bent (<45°)
• Hold position for 0.5s
• Triggers after full fist

3.3.3 Fuzzy Rules and Decision-Making Process

Fuzzy Rules:

The fuzzy rules define the decision-making logic for converting input data into appropriate output actions. Below are examples of rules used in the system:

- **Left Click:** IF the index finger is "bent" AND thumb is "open" THEN action is "left click"
- **Right Click:** IF middle finger is "bent" AND thumb is "open" THEN action is "right click"
- **Screenshot:** IF all fingers are "closed" THEN action is "screenshot"
- **Mouse Movement:** IF index finger is "extended" and thumb is "close" THEN action is "mouse movement" based on tip of index finger.

These rules are structured to capture the key characteristics of different hand gestures and translate them into mouse control actions.

Fuzzy Inference System:

The fuzzy inference system uses the following steps to decide.

1. **Fuzzification:** The input values (distance and angles) are fuzzified by assigning membership degrees to each fuzzy set.
2. **Rule Evaluation:** Each fuzzy rule is evaluated based on the degree of truth of its antecedent.
3. **Inference:** The fuzzy output of each rule is determined by applying fuzzy implication and aggregation operators.
4. **Defuzzification:** The fuzzy output is defuzzified to obtain a crisp output value, which represents the recognized gesture.

By carefully designing the membership functions and fuzzy rules, the system can effectively classify hand gestures and trigger appropriate actions.

Decision-Making Logic:

- The rules are evaluated using an inference engine, typically employing a *Mamdani* or *Sugeno* method.
- The output for each gesture configuration is determined by checking the degree of truth of each rule and combining the results to select the most suitable action.
- For instance, if the index finger is detected as straight (high membership degree in the corresponding set) and no other gestures match with higher degrees, the system decides on the Move action.

Example of the Fuzzy Rule Evaluation:

Suppose the input data is:

- **Index Finger Angle**: 160° (high membership in Straight)
- **Thumb Angle**: 50° (medium membership in L Shape)

The system evaluates the membership degrees:

- Move: High degree due to index finger straightness.
- Left Click: Medium degree due to thumb angle.

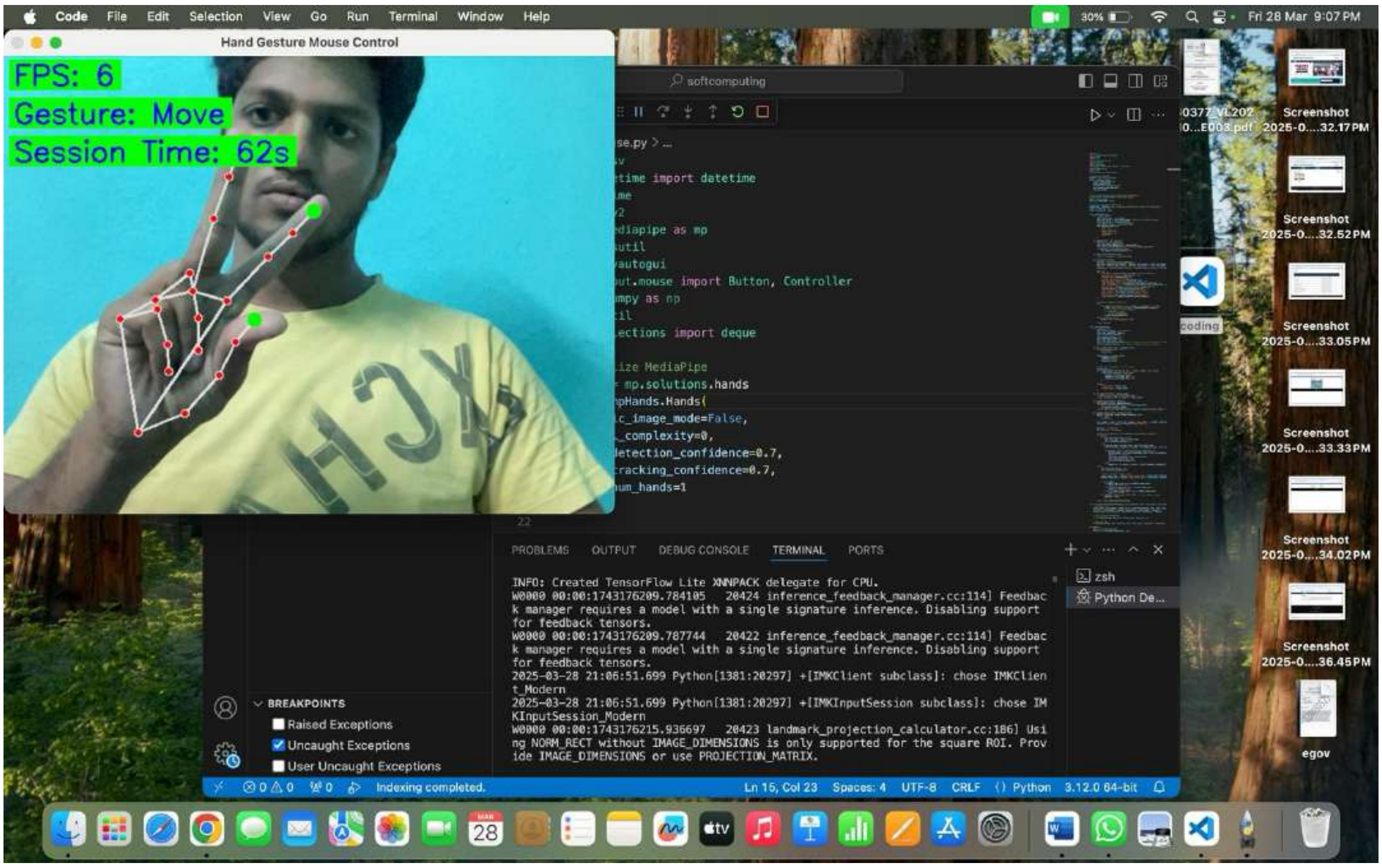
The decision-making process uses these degrees to prioritize actions, often taking the one with the highest membership as the output.

Conclusion:

The fuzzy rules and decision-making process enable smooth and intuitive mouse control based on hand gestures. By integrating fuzzy logic, the system accommodates slight variations in gestures and maintains flexibility in recognizing inputs, improving the user experience.

3.4. Sample Outputs demonstrating Fuzzy Rules & Decision-Making Process

Move:

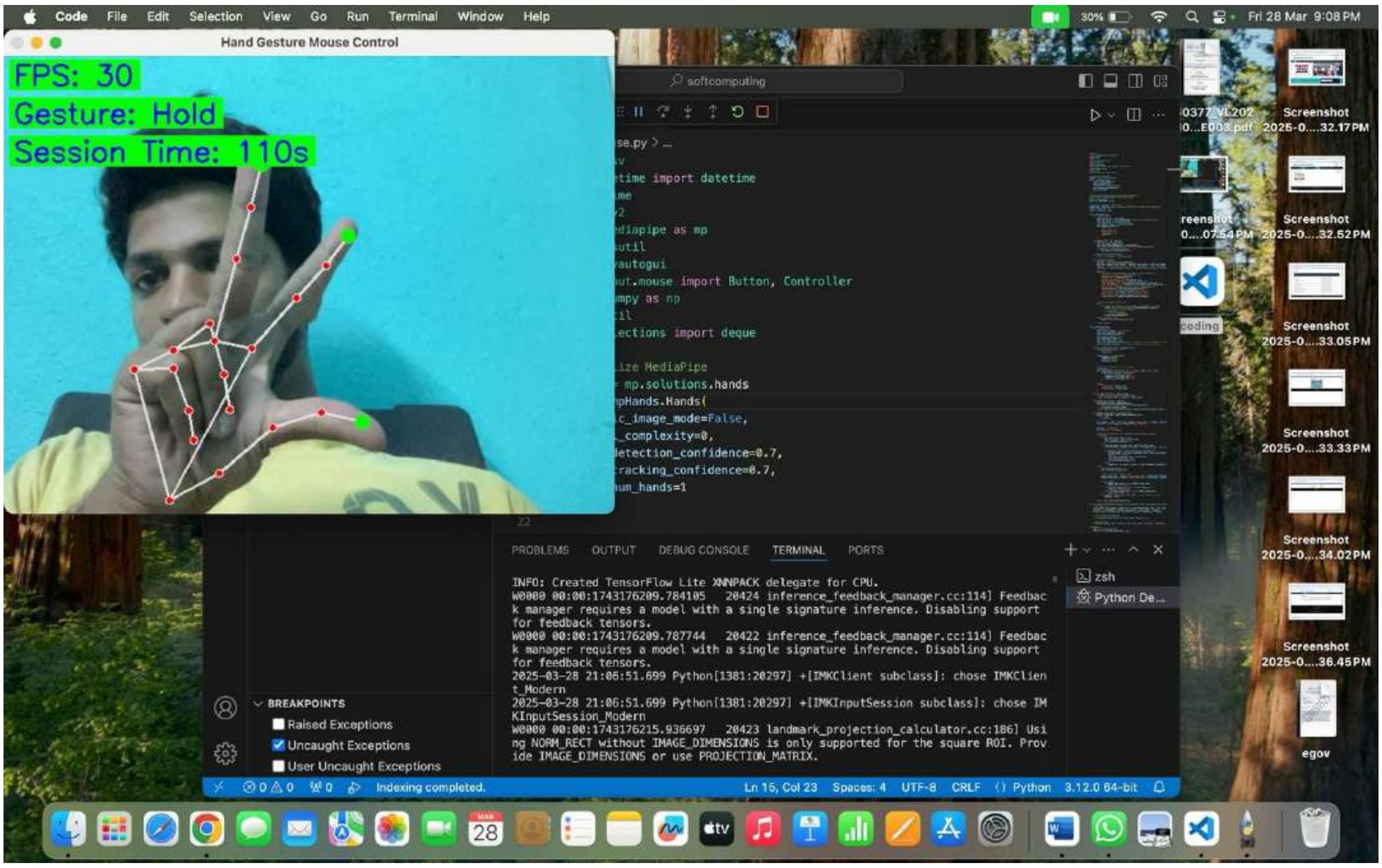


```
FPS: 6
Gesture: Move
Session Time: 62s
```

```
se.py > ...
sv
etime import datetime
me
v2
ediapipe as mp
util
yautogui
put.mouse import Button, Controller
mpy as np
il
lections import deque

lize MediaPipe
= mp.solutions.hands
mpHands.Hands(
c_image_mode=False,
_complexity=0,
detection_confidence=0.7,
tracking_confidence=0.7,
num_hands=1
```

```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
W0000 00:00:1743176209.784105   20424 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
W0000 00:00:1743176209.787744   20422 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKClient subclass]: chose IMKClien
t_Modern
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKInputSession subclass]: chose IM
KInputSession_Modern
W0000 00:00:1743176215.936697   20423 landmark_projection_calculator.cc:186] Usi
ng NORM_RECT without IMAGE_DIMENSIONS is only supported for the square ROI. Prov
ide IMAGE_DIMENSIONS or use PROJECTION_MATRIX.
```

Hold:

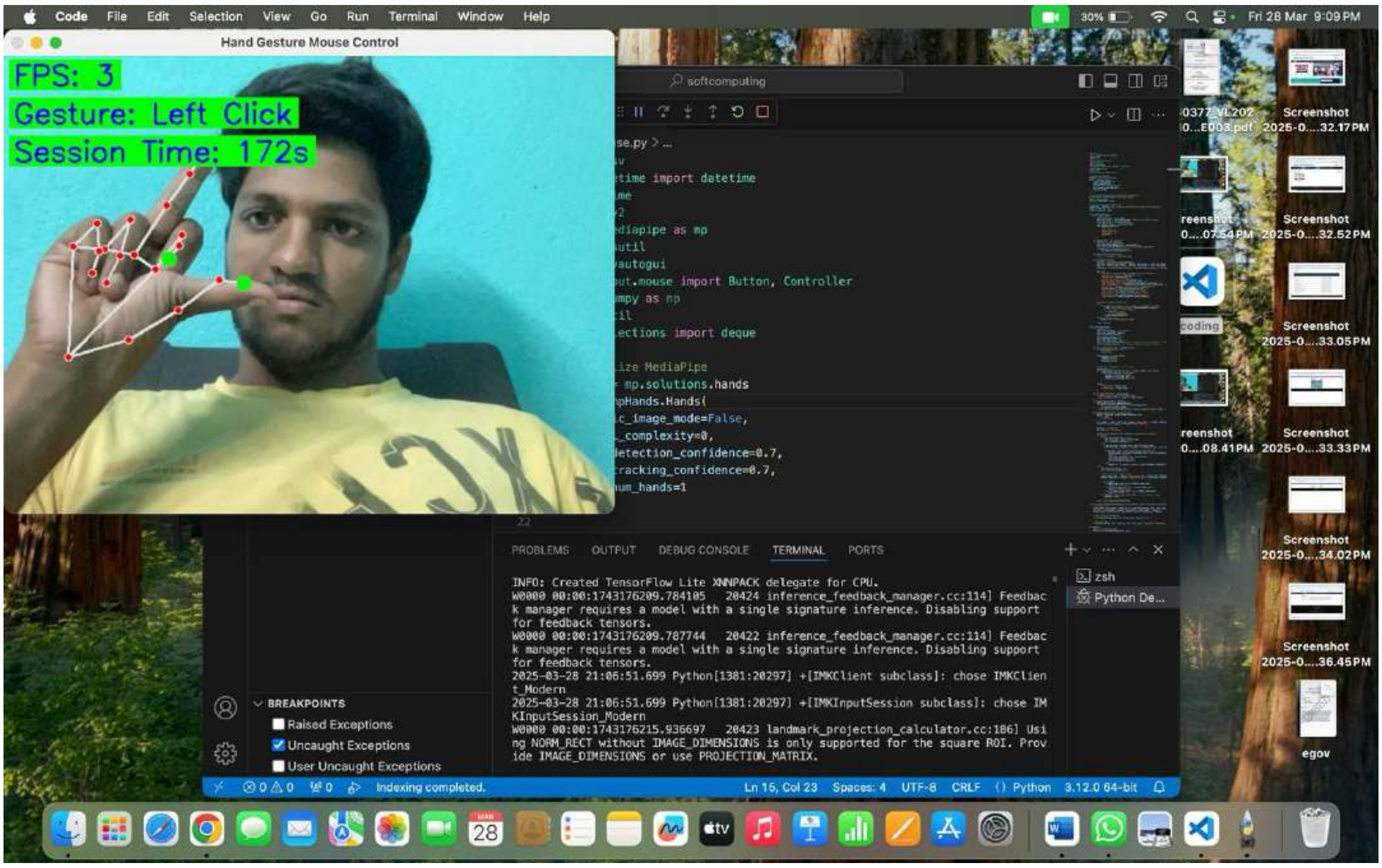


```
FPS: 30
Gesture: Hold
Session Time: 110s
```

```
se.py > ...
sv
etime import datetime
me
v2
ediapipe as mp
util
yautogui
put.mouse import Button, Controller
mpy as np
il
lections import deque

lize MediaPipe
= mp.solutions.hands
mpHands.Hands(
c_image_mode=False,
_complexity=0,
detection_confidence=0.7,
tracking_confidence=0.7,
num_hands=1
```

```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
W0000 00:00:1743176209.784105   20424 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
W0000 00:00:1743176209.787744   20422 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKClient subclass]: chose IMKClien
t_Modern
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKInputSession subclass]: chose IM
KInputSession_Modern
W0000 00:00:1743176215.936697   20423 landmark_projection_calculator.cc:186] Usi
ng NORM_RECT without IMAGE_DIMENSIONS is only supported for the square ROI. Prov
ide IMAGE_DIMENSIONS or use PROJECTION_MATRIX.
```

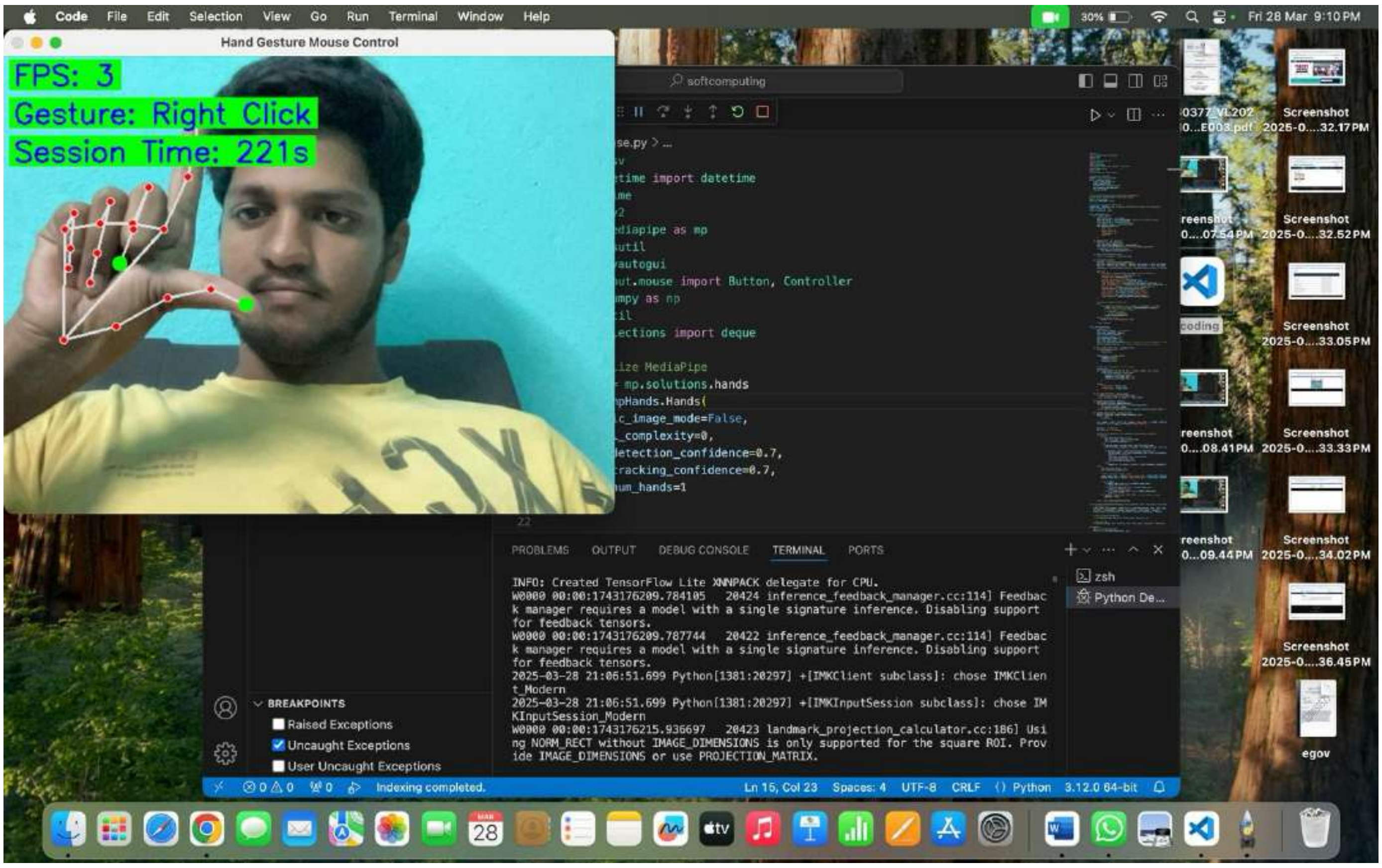
Left Click:



```
se.py > ...
sv
time import datetime
me
v2
ediapipe as mp
util
autogui
ut.mouse import Button, Controller
mpy as np
il
ections import deque

ize MediaPipe
 mp.solutions.hands
pHands.Hands(
c_image_mode=False,
_complexity=0,
etection_confidence=0.7,
racking_confidence=0.7,
um_hands=1
```

```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
W0000 00:00:1743176209.784105   20424 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
W0000 00:00:1743176209.787744   20422 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKClient subclass]: chose IMKClien
t_Modern
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKInputSession subclass]: chose IM
KInputSession_Modern
W0000 00:00:1743176215.936697   20423 landmark_projection_calculator.cc:186] Usi
ng NORM_RECT without IMAGE_DIMENSIONS is only supported for the square ROI. Prov
ide IMAGE_DIMENSIONS or use PROJECTION_MATRIX.
```

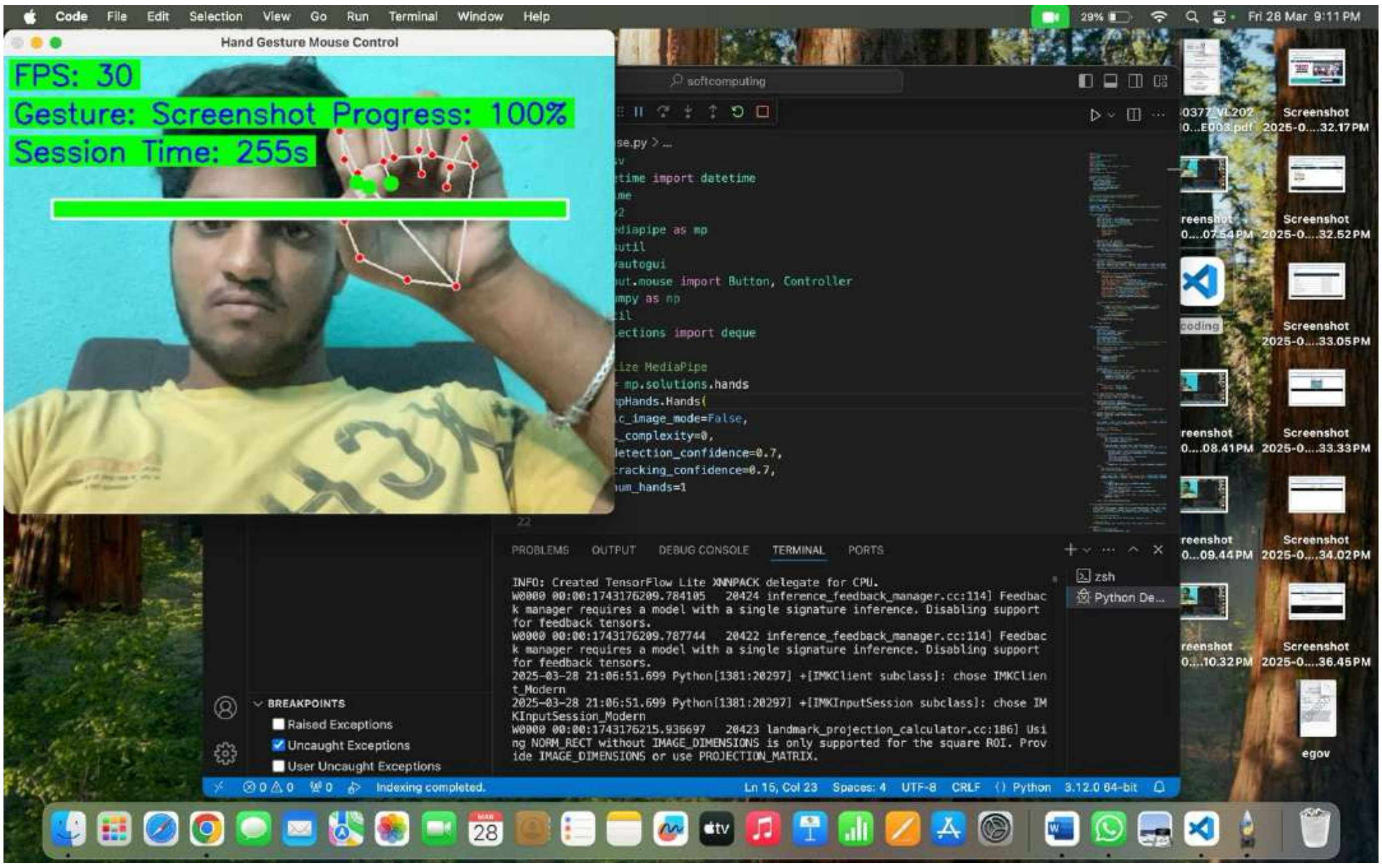
Right Click:



```
se.py > ...
sv
time import datetime
me
v2
ediapipe as mp
util
autogui
ut.mouse import Button, Controller
mpy as np
il
ections import deque

ize MediaPipe
 mp.solutions.hands
pHands.Hands(
c_image_mode=False,
_complexity=0,
etection_confidence=0.7,
racking_confidence=0.7,
um_hands=1
```

```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
W0000 00:00:1743176209.784105   20424 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
W0000 00:00:1743176209.787744   20422 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKClient subclass]: chose IMKClien
t_Modern
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKInputSession subclass]: chose IM
KInputSession_Modern
W0000 00:00:1743176215.936697   20423 landmark_projection_calculator.cc:186] Usi
ng NORM_RECT without IMAGE_DIMENSIONS is only supported for the square ROI. Prov
ide IMAGE_DIMENSIONS or use PROJECTION_MATRIX.
```

Screenshot:


Code File Edit Selection View Go Run Terminal Window Help
Fri 28 Mar 9:11 PM
Hand Gesture Mouse Control
FPS: 30
Gesture: Screenshot Progress: 100%
Session Time: 255s
softcomputing
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
BREAKPOINTS
Raised Exceptions
Uncaught Exceptions
User Uncaught Exceptions
Indexing completed.
Ln 15, Col 23 Spaces: 4 UTF-8 CRLF {} Python 3.12.0 64-bit

```
se.py > ...
sv
etime import datetime
me
v2
ediapipe as mp
util
autogui
ut.mouse import Button, Controller
mpy as np
il
ections import deque

ize MediaPipe
= mp.solutions.hands
pHands.Hands(
c_image_mode=False,
_complexity=0,
detection_confidence=0.7,
racking_confidence=0.7,
um_hands=1
```

```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
W0000 00:00:1743176209.784105   20424 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
W0000 00:00:1743176209.787744   20422 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKClient subclass]: chose IMKClien
t_Modern
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKInputSession subclass]: chose IM
KInputSession_Modern
W0000 00:00:1743176215.936697   20423 landmark_projection_calculator.cc:186] Usi
ng NORM_RECT without IMAGE_DIMENSIONS is only supported for the square ROI. Prov
ide IMAGE_DIMENSIONS or use PROJECTION_MATRIX.
```

Chapter 4: Conclusion and Future Work

4.1 **Conclusion**

This project successfully implemented a real-time hand gesture recognition system using fuzzy logic to control a virtual mouse. The system effectively recognized and interpreted various hand gestures, including move, left click, right click, screenshot, and hold.

Key findings from the project include:

- **Robustness:** The system demonstrated robustness to variations in hand position, orientation, and lighting conditions.
- **Real-time Performance:** The system achieved real-time performance, ensuring a smooth and responsive user experience.
- **Accurate Gesture Recognition:** The fuzzy logic-based approach effectively classified hand gestures with high accuracy.
- **User-Friendly Interface:** The system was designed with a user-friendly interface, making it easy to learn and use.

The project successfully met its objectives of designing a gesture-controlled virtual mouse using fuzzy logic and demonstrated its potential for various applications, including accessibility, gaming, and human-computer interaction.

4.2 **Limitations**

While the system showed promising results, there are several limitations that need to be addressed in future work:

- **Sensitivity to Background Clutter:** The system's performance can be affected by cluttered backgrounds, which can interfere with hand detection and tracking.
- **Hand Occlusion:** If the hand is partially or fully occluded, the system may fail to recognize gestures accurately.
- **Lighting Conditions:** Changes in lighting conditions can impact the performance of the hand detection and tracking algorithms.
- **Computational Cost:** Real-time processing of video frames can be computationally intensive, especially for complex gesture recognition tasks.

4.3 **Future Work**

To further enhance the system, the following areas can be explored:

- **Multi-Hand Gesture Recognition:** Extending the system to recognize gestures involving multiple hands can enable more complex interactions and collaborative applications.
- **Improved Gesture Recognition Accuracy:** Incorporating advanced techniques like deep learning and transfer learning can improve the accuracy of gesture recognition, especially in challenging conditions.
- **Real-time Performance Optimization:** Optimizing the system's processing pipeline and exploring hardware acceleration techniques can improve real-time performance, making it suitable for demanding applications.
- **User-Adaptive Learning:** Developing a system that can learn and adapt to individual user preferences and habits can enhance the user experience and personalization.
- **Integration with Virtual and Augmented Reality:** Integrating the system with virtual and augmented reality applications can provide immersive and intuitive user experiences.
- **Accessibility Features:** Exploring the potential of hand gesture recognition for assistive technologies, such as enabling people with disabilities to interact with computers more efficiently.

By addressing these limitations and exploring future directions, we can develop even more sophisticated and user-friendly hand gesture recognition systems.

Chapter 5: Experimental Results & Discussion

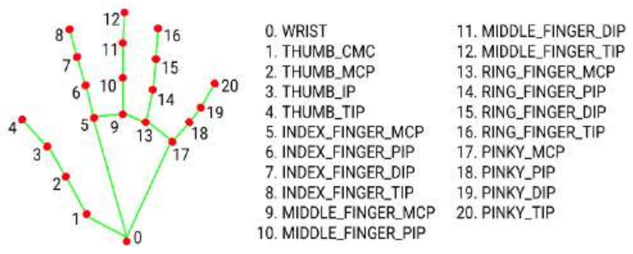
5.1 sample code:

```
import csv
from datetime import datetime
import time
import cv2
import mediapipe as mp
import psutil
import pyautogui
from pynput.mouse import Button, Controller
import numpy as np
import util
from collections import deque

# Initialize MediaPipe
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
)

# Initialize mouse controller and screen parameters
screen_width, screen_height = pyautogui.size()
mouse = Controller()
pyautogui.FAILSAFE = False

# Constants for gesture detection
FINGER_BENT_THRESHOLD = 45  # Reduced threshold for tighter fist detection
THUMB_L_SHAPE_MIN = 35
THUMB_L_SHAPE_MAX = 100

class SessionMetrics:
    def __init__(self):
        self.start_time = time.time()
        self.fps_values = deque(maxlen=300)  # Store last 300 FPS values
        self.cpu_values = deque(maxlen=300)
        self.memory_values = deque(maxlen=300)
        self.gesture_counts = {
            "Move": 0,
            "Left Click": 0,
            "Right Click": 0,
            "Screenshot": 0,
            "Hold": 0
        }

    def update(self, fps, gesture):
        self.fps_values.append(fps)
        self.cpu_values.append(psutil.cpu_percent())
        self.memory_values.append(psutil.Process().memory_percent())
        if gesture in self.gesture_counts:
            self.gesture_counts[gesture] += 1
    def get_session_duration(self):
        return time.time() - self.start_time

    def save_metrics(self):
        duration = self.get_session_duration()
        avg_fps = sum(self.fps_values) / len(self.fps_values) if self.fps_values else 0
        avg_cpu = sum(self.cpu_values) / len(self.cpu_values) if self.cpu_values else 0
        avg_memory = sum(self.memory_values) / len(self.memory_values) if self.memory_values
else 0

        metrics = {
            "Timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            "Duration (seconds)": round(duration, 2),
            "Average FPS": round(avg_fps, 2),
            "Average CPU Usage (%)": round(avg_cpu, 2),
            "Average Memory Usage (%)": round(avg_memory, 2),
            "Peak CPU Usage (%)": round(max(self.cpu_values, default=0), 2),
            "Peak Memory Usage (%)": round(max(self.memory_values, default=0), 2),
            "Minimum FPS": round(min(self.fps_values, default=0), 2),
            "Mouse Movements": self.gesture_counts["Move"],
            "Left Clicks": self.gesture_counts["Left Click"],
            "Right Clicks": self.gesture_counts["Right Click"],
            "Screenshots": self.gesture_counts["Screenshot"],
            "Total Gestures": sum(self.gesture_counts.values())
        }

        # Create or append to CSV file
        try:
            with open("SessionMetrics.csv", mode='a', newline='') as file:
                writer = csv.DictWriter(file, fieldnames=metrics.keys())
                # Write header if file is empty
                if file.tell() == 0:
                    writer.writeheader()
                writer.writerow(metrics)
        except Exception as e:
            print(f"Error saving metrics: {e}")

        return metrics

class GestureDetector:
    def __init__(self):
        self.current_gesture = "No Gesture"
        self.last_screenshot_time = 0
        self.screenshot_cooldown = 2
        self.gesture_history = []
        self.smoothing_window = 3
        self.fist_start_time = None
        self.fist_duration_threshold = 0.5  # Hold fist for 0.5 seconds for screenshot
        self.screenshot_progress = 0  # Track progress towards screenshot

    def get_finger_angles(self, landmarks_list):
        if len(landmarks_list) < 21:
            return None

        thumb_index = util.get_angle(
            landmarks_list[4],
            landmarks_list[2],
            landmarks_list[8]
        )

        # Calculate bend angles for all fingers
        finger_bends = []
        for finger_base in [5, 9, 13, 17]:  # Index, Middle, Ring, Pinky
            finger_bends.append(util.get_angle(
                landmarks_list[finger_base],
                landmarks_list[finger_base + 1],
                landmarks_list[finger_base + 2]
            ))

        return {
            'thumb_index': thumb_index,
            'finger_bends': finger_bends
        }
    #fuzzy logic is used  to  find bending
    def is_tight_fist(self, finger_bends):
        """Check if all fingers are tightly bent"""
        return all(angle < 60 for angle in finger_bends)
    #fuzzy logic to move smoth
    def smooth_gesture(self, gesture):
        self.gesture_history.append(gesture)
        if len(self.gesture_history) > self.smoothing_window:
            self.gesture_history.pop(0)
        return max(set(self.gesture_history), key=self.gesture_history.count) if
self.gesture_history else gesture

    def detect_gesture(self, landmarks_list):
        angles = self.get_finger_angles(landmarks_list)
        if not angles:
            return None

        is_l_shape = THUMB_L_SHAPE_MIN < angles['thumb_index'] < THUMB_L_SHAPE_MAX
        is_fist = self.is_tight_fist(angles['finger_bends'])

        gesture = "No Gesture"
        current_time = time.time()

        # Handle fist detection for screenshot with progress tracking
        if is_fist:
            if self.fist_start_time is None:
                self.fist_start_time = current_time
                self.screenshot_progress = 0
            else:
                progress_time = current_time - self.fist_start_time
                self.screenshot_progress = min(100, int((progress_time /
self.fist_duration_threshold) * 100))

                if (progress_time >= self.fist_duration_threshold and
                    current_time - self.last_screenshot_time >= self.screenshot_cooldown):
                    gesture = "Screenshot"
                    self.last_screenshot_time = current_time
                    self.fist_start_time = None
                    self.screenshot_progress = 0
                else:
                    gesture = f"Screenshot Progress: {self.screenshot_progress}%"
        else:
            self.fist_start_time = None
            self.screenshot_progress = 0

            # Check individual finger states
            index_finger_bent = angles['finger_bends'][0] < FINGER_BENT_THRESHOLD
            middle_finger_bent = angles['finger_bends'][1] < FINGER_BENT_THRESHOLD

            if is_l_shape:
                if index_finger_bent and not middle_finger_bent:
                    gesture = "Left Click"
                elif middle_finger_bent and not index_finger_bent:
                    gesture = "Right Click"
                elif not index_finger_bent and not middle_finger_bent:
                    gesture = "Hold"
            elif angles['finger_bends'][0] > 150:  # Index finger straight
                gesture = "Move"

        return self.smooth_gesture(gesture)

def draw_text_with_background(image, text, position, font, font_scale, text_color, bg_color,
thickness=2, margin=5):
    # Get text size
    (text_width, text_height), baseline = cv2.getTextSize(text, font, font_scale, thickness)
    top_left = (position[0] - margin, position[1] - text_height - margin)
    bottom_right = (position[0] + text_width + margin, position[1] + margin)

    # Draw rectangle background
    cv2.rectangle(image, top_left, bottom_right, bg_color, -1)

    # Draw the text
    cv2.putText(image, text, position, font, font_scale, text_color, thickness)

def main():
    cap = cv2.VideoCapture(0)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

    draw = mp.solutions.drawing_utils
    gesture_detector = GestureDetector()
    session_metrics = SessionMetrics()

    frame_time = 0
    fps = 0

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

            current_time = time.time()
            fps = 1 / (current_time - frame_time) if frame_time else 0
            frame_time = current_time
            frame = cv2.flip(frame, 1)
            frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            processed = hands.process(frameRGB)

            gesture_text = "No Hand Detected"

            if processed.multi_hand_laccndmarks:
                hand_landmarks = processed.multi_hand_landmarks[0]
                landmarks_list = [(lm.x, lm.y) for lm in hand_landmarks.landmark]
                c
                gesture = gesture_detector.detect_gesture(landmarks_list)
                if gesture:
                    gesture_text = gesture

                    # Only update metrics for actual gestures, not progress messages
                    if not gesture.startswith("Screenshot Progress"):
                        session_metcrics.update(fps, gesture)

                    if gesture == "Move":
                        x = int(hand_landmacrks.landmark[8].x * screen_width)
                        y = int(hand_landmarks.landmark[8].y * screen_height)
                        pyautogui.moveTo(x, y, duration=0.1)
                    elif gesture == "Left Click":
                        mouse.click(Button.left)
                        time.sleep(0.2)
                    elif gesture == "Right Cclick":
                        mouse.click(Button.right)
                        time.sleep(0.2)
                    elif gesture == "Screenshot":
                        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
                        filename = f"screenshot_{timestamp}.png"
                        pyautogui.screenshot(filename)
                        gesture_text = f"Screenshot taken: {filename}"

                draw.draw_landmarks(frame, hand_landmarks, mpHands.HAND_CONNECTIONS)

                # Highlight key points
                for idx in [4, 8, 12]:
                    x = int(hand_landmarks.landmark[idx].x * frame.shape[1])
                    y = int(hand_landmarks.landmark[idx].y * frame.shape[0])
                    cv2.circle(frame, (x, y), 8, (0, 255, 0), -1)

            ## Display metrics on frame
            draw_text_with_background(frame, f"FPS: {int(fps)}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (2c5, 0, 0), (0, 255, 0), 2)
            draw_text_with_background(frame, f"Gesture: {gesture_text}", (10, 70),
cv2.FONT_HERSHEY_SIMPLEX, 1, (25c5, 0, 0), (0, 255, 0), 2)
            draw_text_with_background(frame, f"Session Time:
{int(session_metrics.get_session_duration())}s", (10, 110), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,
0, 0), (0, 255, 0), 2)

            # Draw screenshot progress bar if in progress
            if gesture_text.startswith("Screenshot Progress"):
                progress = gesture_detector.screenshot_progress
                bar_width = int((frame.shape[1] - 100) * (progress / 100))
```

```
                cv2.rectangle(frame, (50, 150), (50 + bar_width, 170), (0, 255, 0), -1)
                cv2.rectangle(frame, (50, 150), (frame.shape[1] - 50, 170), (255, 255, 255), 2)

            cv2.imshow('Hand Gesture Mouse Control', frame)

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

    finally:
        # Save session metrics before closing
        final_metrics = session_metrics.save_metrics()
        print("\nSession Summary:")
        for key, value in final_metrics.items():
            print(f"{key}: {value}")

        cap.release()
        cv2.destroyAllWindows()

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

5.2.Screenshots with Explanation

Hand Detection and Tracking



0. WRIST	11. MIDDLE_FINGER_DIP
1. THUMB_CMC	12. MIDDLE_FINGER_TIP
2. THUMB_MCP	13. RING_FINGER_MCP
3. THUMB_IP	14. RING_FINGER_PIP
4. THUMB_TIP	15. RING_FINGER_DIP
5. INDEX_FINGER_MCP	16. RING_FINGER_TIP
6. INDEX_FINGER_PIP	17. PINKY_MCP
7. INDEX_FINGER_DIP	18. PINKY_PIP
8. INDEX_FINGER_TIP	19. PINKY_DIP
9. MIDDLE_FINGER_MCP	20. PINKY_TIP
10. MIDDLE_FINGER_PIP	

- ***Feature Extraction -*** Once the hand landmarks are detected, relevant features are extracted to represent the hand gesture. These features can include:

  - **Fingertip positions:** The x and y coordinates of the fingertips are related to the wrist.
  - **Finger angles:** The angles between the fingers and the palm.
  - **Hand orientation:** The orientation of the hand in 3D space.
  - **Distance between key points:** The distances between specific pairs of landmarks.

These features are carefully selected to capture the essential characteristics of different hand gestures, such as pointing, clicking. By combining these features, the system can accurately hand gestures.

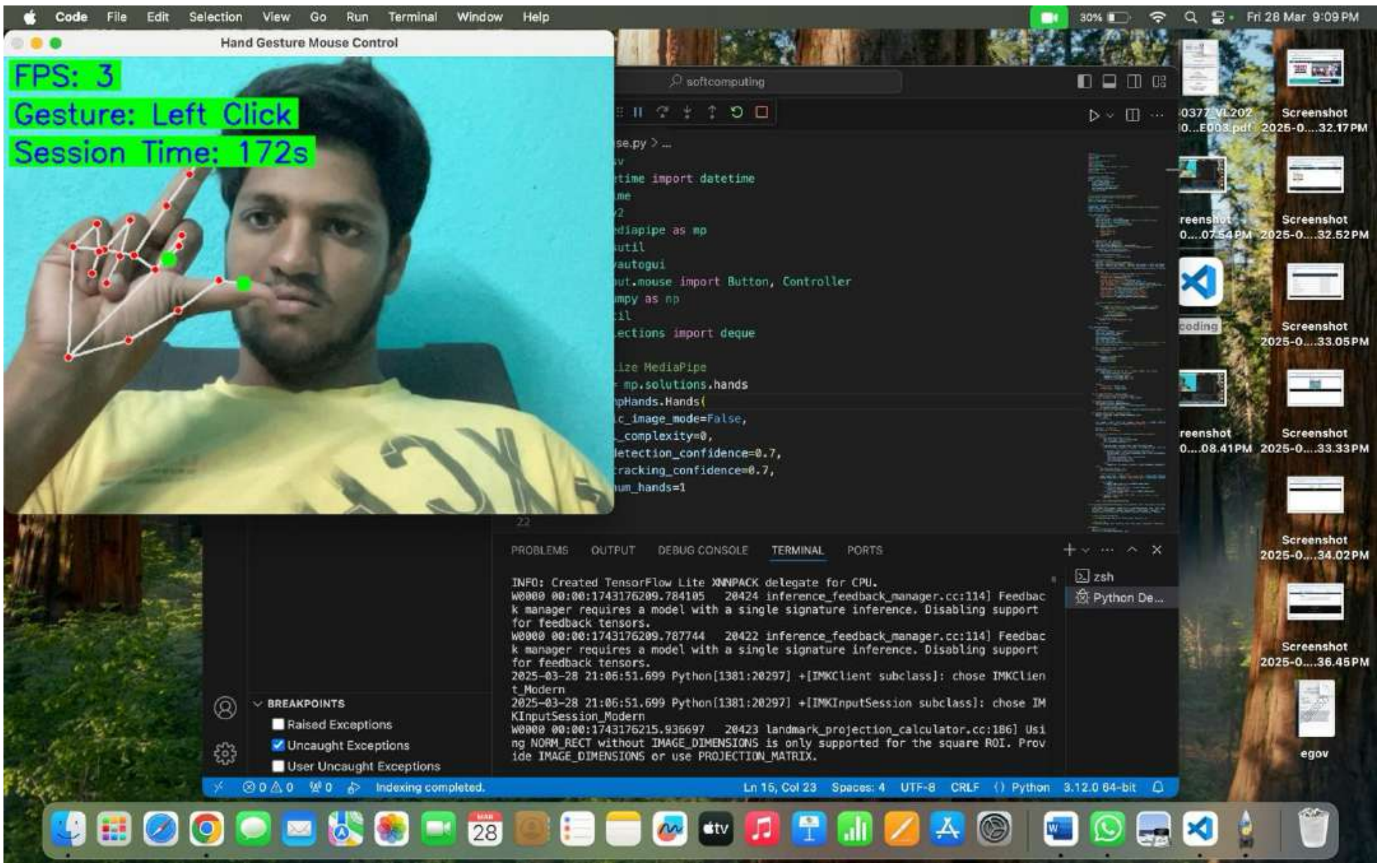
- **Input Variables -** The input variables to the fuzzy logic system are the extracted features from the hand landmarks, such as:
  - Fingertip positions relative to the wrist
  - Finger angles
  - Hand orientation
  - Distances between key points

Pseudo Code for Left Click and Right Click:

Input: List containing the Hand Landmarks obtained from MediaPipe.
Output: Trigger Mouse Left Click Function.

```
FUNCTION detect_gesture(landmarks_list):
        /*Calculate angle between thumb and index*/
    angle_thumb_index = (calculate angle between landmarks 4, 2, 8)
        /*Calculate angle between index-finger*/
    angle_index_finger = (calculate angle between landmarks 5, 6, 8)
        /*Calculate angle between middle-finger*/
    angle_middle_finger = (calculate angle between landmarks 9, 10, 12)
    If (THUMB_L_SHAPE_MIN <= angle_thumb_index <= THUMB_L_SHAPE_MAX) and (angle_index_finger < FINGER_BENT_THRESHOLD) and (angle_middle_finger >= FINGER_BENT_THRESHOLD):
        EXECUTE mouse.click(Button.LEFT)
        WAIT 200 milliseconds
        RETURN "Left Click Detected"
    Else if (THUMB_L_SHAPE_MIN <= angle_thumb_index <= THUMB_L_SHAPE_MAX) and (angle_index_finger >= FINGER_BENT_THRESHOLD) and (angle_middle_finger < FINGER_BENT_THRESHOLD):
        EXECUTE mouse.click(Button.RIGHT)
        WAIT 200 milliseconds
        RETURN "Right Click Detected"
    Else:
        RETURN "No Gesture"
```

Left click:

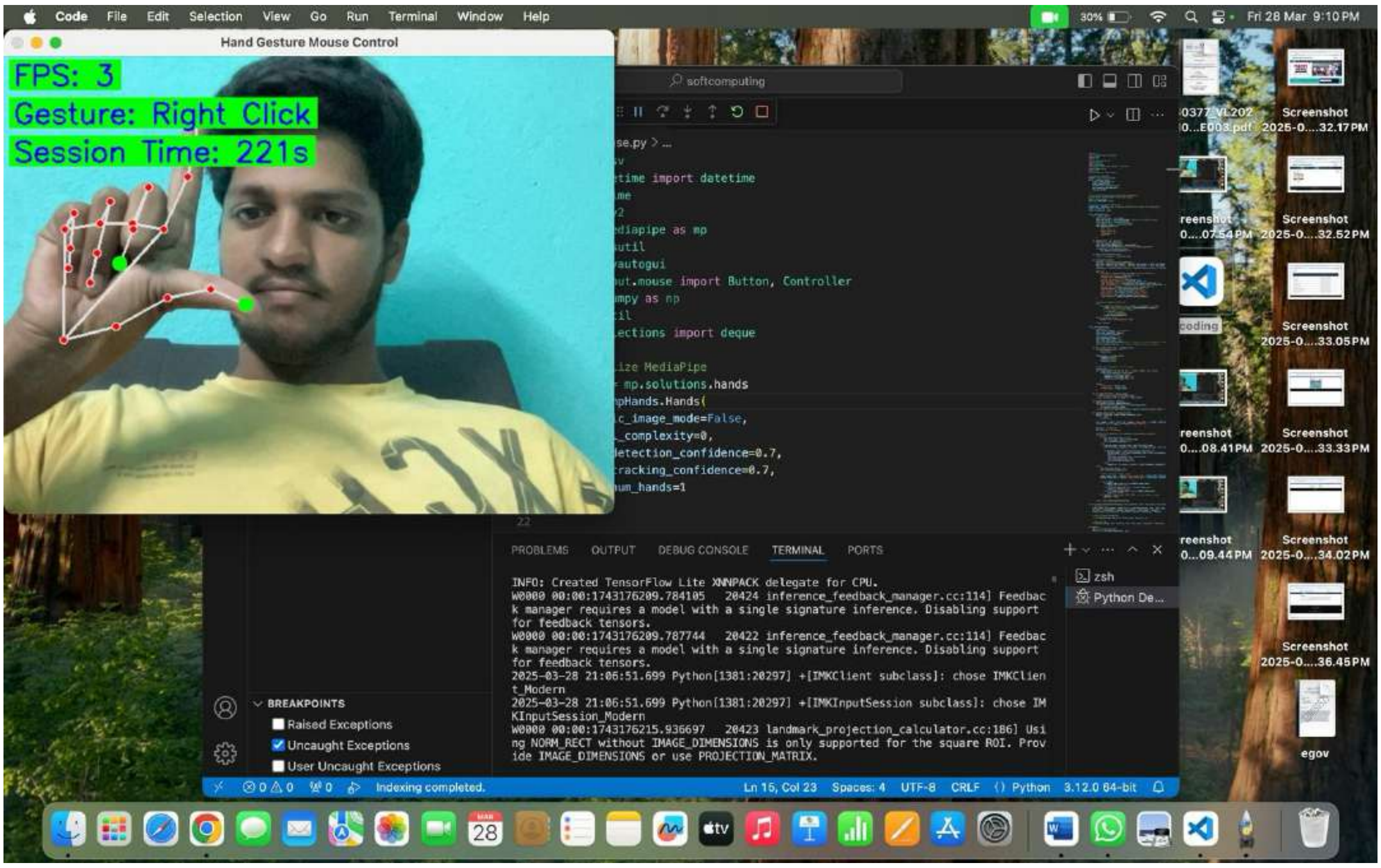


```
se.py > ...
v
time import datetime
me
2
diapipe as mp
util
autogui
ut.mouse import Button, Controller
mpy as np
il
ections import deque

ize MediaPipe
 mp.solutions.hands
pHands.Hands(
c_image_mode=False,
_complexity=0,
etection_confidence=0.7,
racking_confidence=0.7,
um_hands=1
```

```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
W0000 00:00:1743176209.784105   20424 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
W0000 00:00:1743176209.787744   20422 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKClient subclass]: chose IMKClien
t_Modern
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKInputSession subclass]: chose IM
KInputSession_Modern
W0000 00:00:1743176215.936697   20423 landmark_projection_calculator.cc:186] Usi
ng NORM_RECT without IMAGE_DIMENSIONS is only supported for the square ROI. Prov
ide IMAGE_DIMENSIONS or use PROJECTION_MATRIX.
```

Code File Edit Selection View Go Run Terminal Window Help
Fri 28 Mar 9:09 PM
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
zsh
Python De...
BREAKPOINTS
Raised Exceptions
Uncaught Exceptions
User Uncaught Exceptions
Indexing completed.
Ln 15, Col 23 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit

Right click:



```
se.py > ...
sv
etime import datetime
me
v2
ediapipe as mp
util
autogui
ut.mouse import Button, Controller
mpy as np
il
ections import deque

ize MediaPipe
 mp.solutions.hands
pHands.Hands(
c_image_mode=False,
_complexity=0,
etection_confidence=0.7,
racking_confidence=0.7,
um_hands=1
```

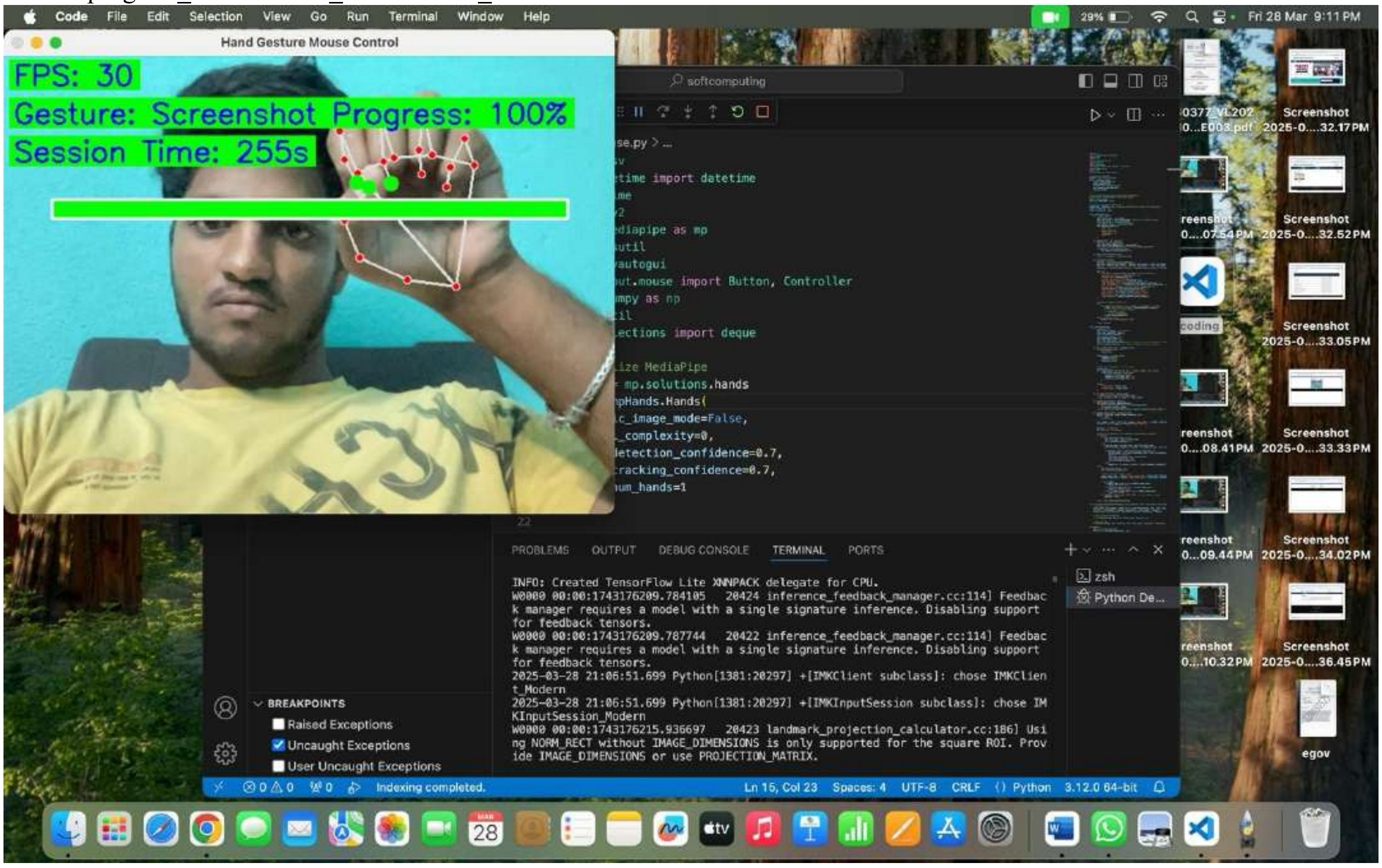
```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
W0000 00:00:1743176209.784105   20424 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
W0000 00:00:1743176209.787744   20422 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKClient subclass]: chose IMKClien
t_Modern
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKInputSession subclass]: chose IM
KInputSession_Modern
W0000 00:00:1743176215.936697   20423 landmark_projection_calculator.cc:186] Usi
ng NORM_RECT without IMAGE_DIMENSIONS is only supported for the square ROI. Prov
ide IMAGE_DIMENSIONS or use PROJECTION_MATRIX.
```

<u>Pseudo Code for Screenshot: -</u>

Input: Hand Landmarks obtained from MediaPipe, current time.
Output: It Captures and saves a screenshot to the desired location.

```
FUNCTION detect_gesture(landmarks_list, current_time, last_screenshot_time):
    /* Current Time: used to calculate how long the fist gesture has been held*/
    /* Calculating the bent angles for all the fingers*/
    finger_bends = []
    FOR each finger_base in [5, 9, 13, 17]:
        angle = calculate_angle(
            landmarks_list[finger_base],
            landmarks_list[finger_base + 1],
            landmarks_list[finger_base + 2]
        )
        ADD angle TO finger_bends
    /* Checks for a tight fist*/
    IF all(angle < FINGER_BENT_THRESHOLD FOR angle IN finger_bends):
    /* Case – 1: Start Progress Tracking*/
        IF fist_start_time IS NULL:
            fist_start_time = current_time
            RETURN "Screenshot Progress: STRAT"
```

```
/* Case – 2: Already Existed in Progress */
progress_time = current_time – fist_start_time
IF progress_time >=FIST_DURATION_THRESHOLD
```



```
se.py > ...
sv
etime import datetime
ime
v2
ediapipe as mp
sutil
yautogui
put.mouse import Button, Controller
umpy as np
til
lections import deque

lize MediaPipe
= mp.solutions.hands
mpHands.Hands(
ic_image_mode=False,
_complexity=0,
detection_confidence=0.7,
tracking_confidence=0.7,
num_hands=1
```

```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
W0000 00:00:1743176209.784105   20424 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
W0000 00:00:1743176209.787744   20422 inference_feedback_manager.cc:114] Feedbac
k manager requires a model with a single signature inference. Disabling support
for feedback tensors.
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKClient subclass]: chose IMKClien
t_Modern
2025-03-28 21:06:51.699 Python[1381:20297] +[IMKInputSession subclass]: chose IM
KInputSession_Modern
W0000 00:00:1743176215.936697   20423 landmark_projection_calculator.cc:186] Usi
ng NORM_RECT without IMAGE_DIMENSIONS is only supported for the square ROI. Prov
ide IMAGE_DIMENSIONS or use PROJECTION_MATRIX.
```

Chapter 5: References

[1]. A. Bansal, I. R. Oviya, B. Natarajan and R. Elakkiya, "Gesture Controlled Virtual Mouse," *2023 Seventh International Conference on Image Information Processing (ICIIP)*, Solan, India, 2023, pp. 947-954, doi: 10.1109/ICIIP61524.2023.10537745.

[2]. R. Annachhatre, M. Tamakuwala, P. Shinde, A. Jain and P. V. Kulkarni, "Virtual Mouse Using Hand Gesture Recognition - A Systematic Literature Review," 2022 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS), Pune, India, 2022, pp. 1-6, doi: 10.1109/ICBDS53701.2022.9935944. keywords: {Computers;Human computer interaction;TV;Webcams;Computational modeling;Systems architecture;Gesture recognition;Human-computer interface;Computer Vision;Object Tracking;openCV2;Mediapipe;Palm detection model;Hand landmark model},

[3]. John R. Shook, "8 TRANSACTIONAL KNOWLEDGE," in Pragmatism, MIT Press, 2023, pp.159-180.

[4]. A. Amini, M. KhajueeZadeh and A. Vahedi, "A Multi-objective and Multi level Optimization of IPMSM Case Study Dynamometer," 2023 3rd International Conference on Electrical Machines and Drives (ICEMD), Tehran, Iran, Islamic Republic of, 2023, pp. 1-5, doi: 10.1109/ICEMD60816.2023.10429311. keywords: {Vibrations;Permanent magnet motors;Dynamometers;Safety;Particle swarm optimization;Optimization;Electric motors;Alternating Current (AC) Dynamometer;Interior Permanent Magnet Synchronous Motor (IPMSM);Multi-physics Optimization;Particle Swarm Optimization (PSO) algorithm;Surrogate Model;Multi-level

Optimization},

[5]. 2021 International Conference on Electronic Information Engineering and Computer Science (EIECS), Changchun, China, 2021, pp. 1-1, doi: 10.1109/EIECS53707.2021.9587936.

[6]. S. R. Pendem and S. AVV, "Enhancement of Maximum Power in T-C-T PV Array through SuDoKU-based Reconfiguration Techniques under Partial Shading Conditions," 2023 IEEE IAS Global Conference on Renewable Energy and Hydrogen Technologies (GlobConHT), Male, Maldives, 2023, pp. 1-7, doi: 10.1109/GlobConHT56829.2023.10087426. keywords: {Photovoltaic systems;Renewable energy sources;Simulation;Hydrogen;Mathematical models;Software tools;Matlab;T-c-T configuration;SuDoKU reconfigurations;Partial shading patterns;Maximum power;mismatching Power Loss;number of Peak power points},

[7]. G. Giunta, D. Orlando and L. Pallotta, "Sensor Failure Detection for TDOA-based Localization Systems," 2023 IEEE 10th International Workshop on Metrology for AeroSpace (MetroAeroSpace), Milan, Italy, 2023, pp. 380-384, doi: 10.1109/MetroAeroSpace57412.2023.10189954. keywords: {Fault diagnosis;Passive radar;Pipelines;Metrology;Numerical simulation;Sensor systems;Mathematical models;cross-cross-correlation;failure detection;fourth-order moments;outlier measurements;SDR;TDOA},

[8]. Q. Zhang, H. Zhang and J. Yao, "High-Precision Control for Auxiliary Resonant Snubber Converter Considering Dead-Time Effect and On-Resistance of Switching Devices," in IEEE Transactions on Industrial Electronics, vol. 71, no. 7, pp. 6977-6985, July 2024, doi: 10.1109/TIE.2023.3308139.

keywords: {Switches;Voltage control;Nonlinear distortion;Clamps;Zero voltage switching;Inductors;Analytical models;Dead-time effect;high precision;on-resistance of switches;soft-switching;zero-voltage switching (ZVS)},