



Department of Computer Engineering

Software Engineering Project Report

“Real-Time Hand Gesture Recognition and Control System using Python”

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AISSMS IOIT

S. Y. BTech Computer Engineering

SUBMITTED BY

Roll No.

Exam Seat Number

Name

89

SY2304082

Harsh R. Meshram



2023-24



Department of Computer Engineering

CERTIFICATE

This is to certify that the project report

“ Real-Time Hand Gesture Recognition and Control System
using Python ”

Submitted by

Student Roll No	Student Name
89	Harsh R. Meshram

is a bonafide student of this institute and the work has been carried out by her under the supervision of **Dr. Veena Bhende** and it is approved for the partial fulfillment of the Department of Computer Engineering AISSMS IOIT.



(Dr. Veena Bhende)

Guide

Department of Computer Engineering
Place : Pune

(Dr. S. N. Zaware)

Head,

Department of Computer Engineering
Date :



Index

Sr.no	Chapter No.	Title of Chapter	Page no.
1.	1	INTRODUCTION	1-6
	1.1	Title	5
	1.2	Objective	5
	1.3	Features of Project	6
2.	2	DESIGN AND ARCHITECTURE	7-10
	2.1	Data Center Architecture	7
	2.2	Data Flow Architecture	8-9
	2.3	Call and Return Architecture	10
3.	3	UML DIAGRAM	11-14
	3.1	Use Case Diagram	11
	3.2	Class Diagram	11
	3.3	Sequence Diagram	12
	3.4	Collaboration Diagram	12
	3.5	Activity Diagram	13
	3.6	Component Diagram	14



	3.7	Deployment Diagram	14
4.	4	SOFTWARE METRICS	15-17
	4.1	Methods to find cost of software	15
	4.2	Method applied for calculating the cost of the system.	16-17
5.	5	SOFTWARE TESTING	18-21
	5.1	Black Box testing on any GUI	18-19
	5.2	White Box testing of any running code.	19-21
6.	6	CODE AND OUTPUT	22-34
	6.1	Running Code	22-30
	6.2	Output of the project	31-34
7	7	CONCLUSION AND REFERENCES	35
	7.1	Conclusion	35
	7.2	References	35

CHAPTER 1: INTRODUCTION

1.1 TITLE

The title of my project is “**Real-Time Hand Gesture Recognition and Control System using Python**” enables intuitive computer interaction by tracking hand gestures and translating them into mouse movements and clicks.

1.2 OBJECTIVE

The objective of this project is to develop an advanced gesture recognition system that enables users to control their computer through intuitive hand gestures captured by a webcam. Utilizing cutting-edge computer vision techniques and machine learning models for hand tracking and gesture classification, the system aims to interpret specific hand movements and translate them into practical commands for system operations. This includes functionalities such as moving the mouse cursor, clicking, scrolling, adjusting volume, and controlling screen brightness. The project seeks to enhance user interaction by providing a hands-free, real-time, and user-friendly interface, thereby improving accessibility and offering an innovative method of computer control for various applications.

1.3 FEATURES OF THE PROJECT

The features of the “Real-Time Hand Gesture Recognition and Control System using Python” project are:

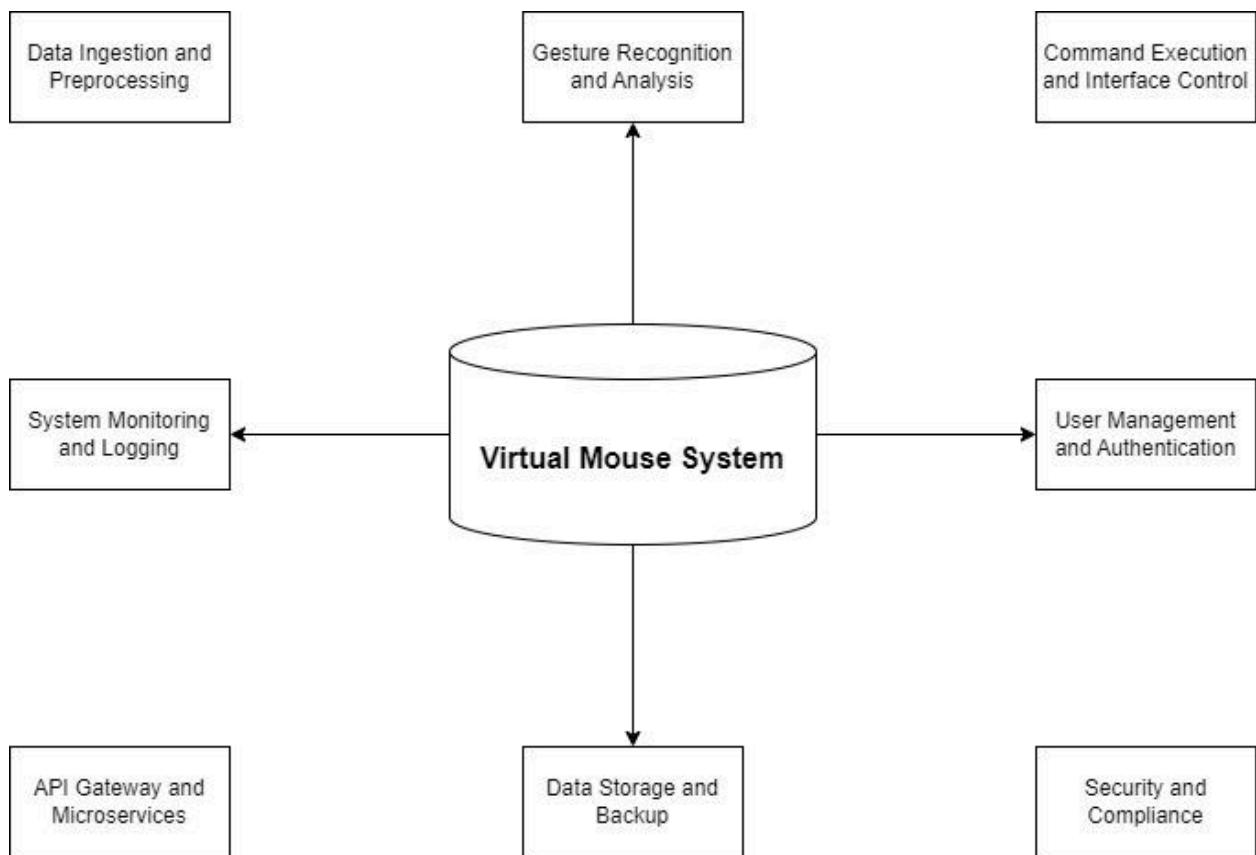
1. **Real-time Hand Gesture Recognition:** Utilizes AI algorithms to accurately detect and interpret hand gestures in real-time.

2. **Intuitive Cursor Control:** Enables users to control the mouse cursor using hand movements, providing a natural and intuitive interaction experience.
3. **Customizable Gestures:** Offers the flexibility for users to customize hand gestures for specific mouse actions, allowing for personalized control preferences.
4. **Adaptive Sensitivity:** Adjusts cursor movement sensitivity dynamically based on hand motion, ensuring precise control without being overly sensitive or unresponsive.
5. **Cross-Platform Compatibility:** Supports multiple operating systems such as Windows, macOS, and Linux, making it accessible to a wide range of users across different platforms.
6. **Accessibility Features:** Incorporates accessibility features to cater to users with disabilities, such as adjustable cursor speed and voice commands for mouse actions.
7. **Gesture Feedback:** Provides visual and/or auditory feedback to users upon successful recognition of hand gestures, enhancing the user experience and confirming input actions.
8. **Low Latency Interaction:** Minimizes latency between hand gestures and cursor movement, delivering a responsive and seamless interaction between the user and the virtual mouse.
9. **Robust Error Handling:** Implements error detection and recovery mechanisms to handle unexpected scenarios, ensuring the stability and reliability of the virtual mouse system.
10. **Open-Source Development:** Offers the project as open-source software, allowing for community contributions, feedback, and continuous improvement of the virtual mouse functionality.



CHAPTER 2: DESIGN AND ARCHITECTURE

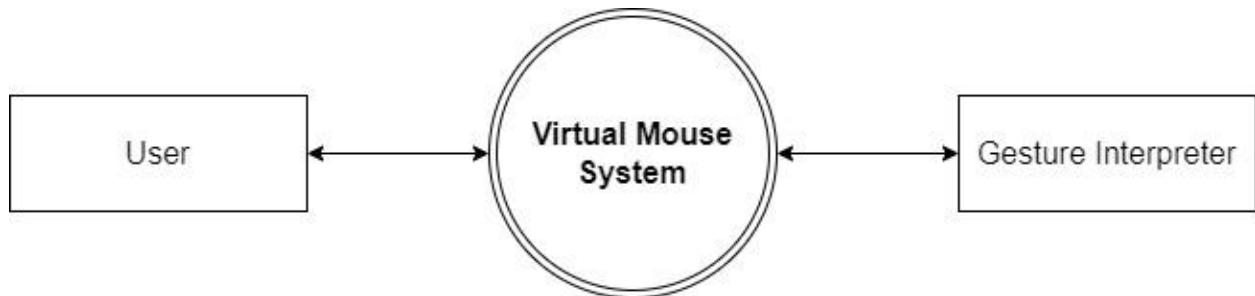
2.1 DATA CENTER ARCHITECTURE



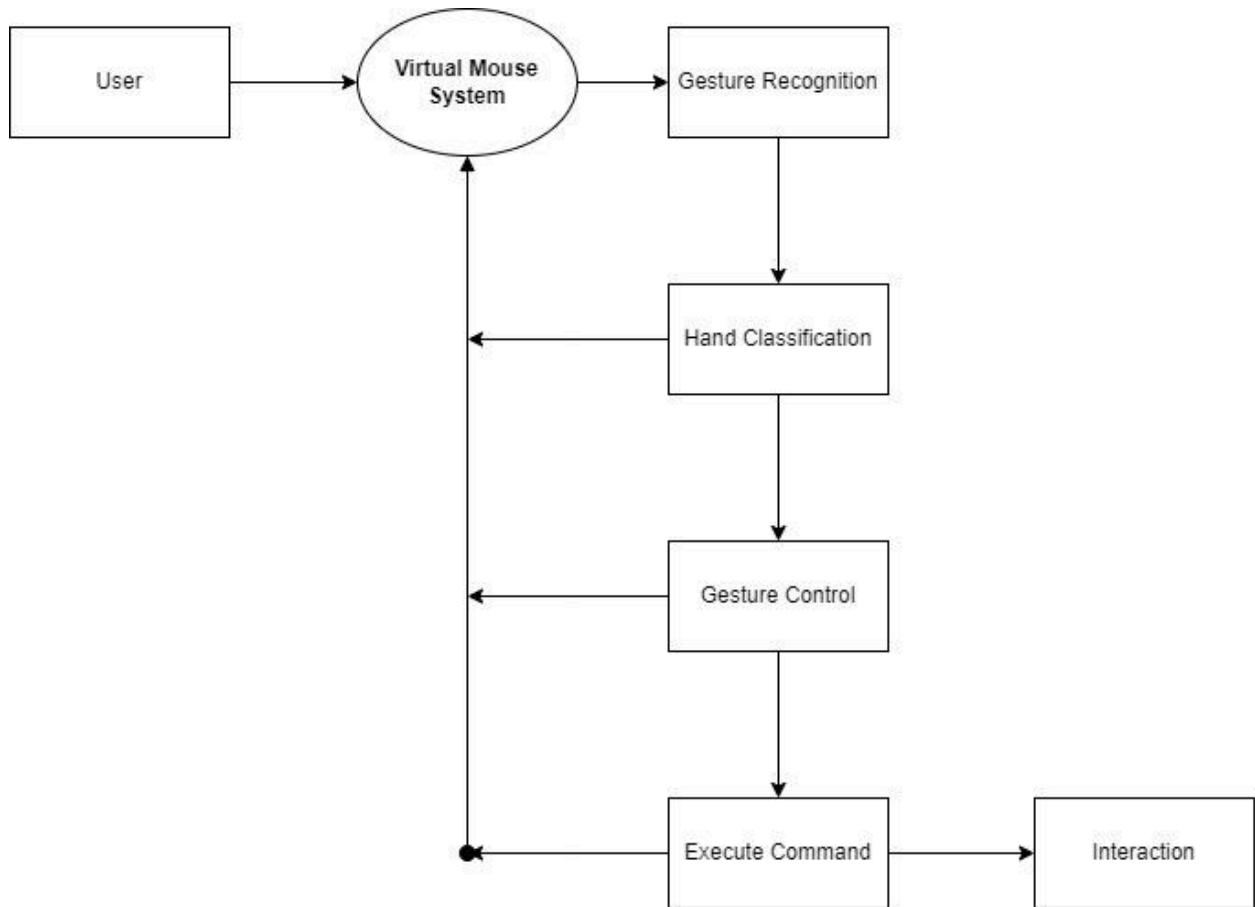


2.2 DATA FLOW ARCHITECTURE

2.2.1 DFD LEVEL 0:

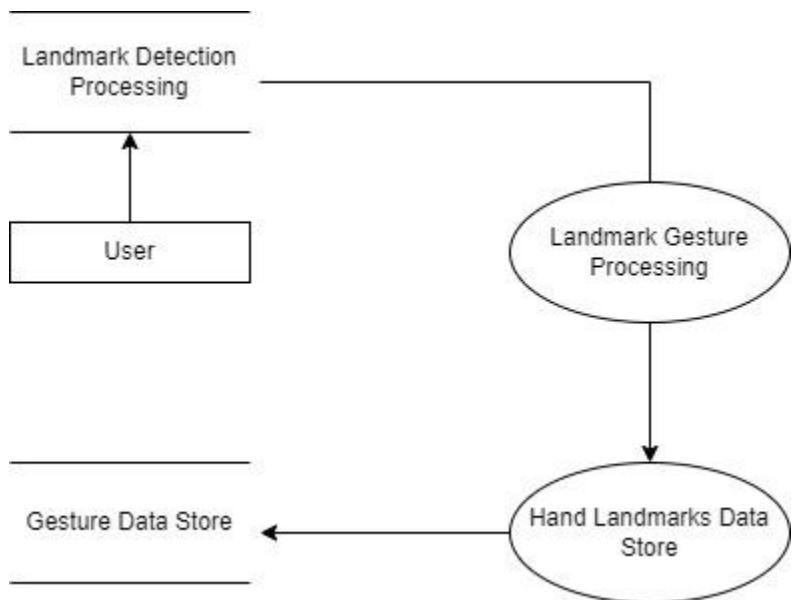


2.2.2 DFD LEVEL 1:



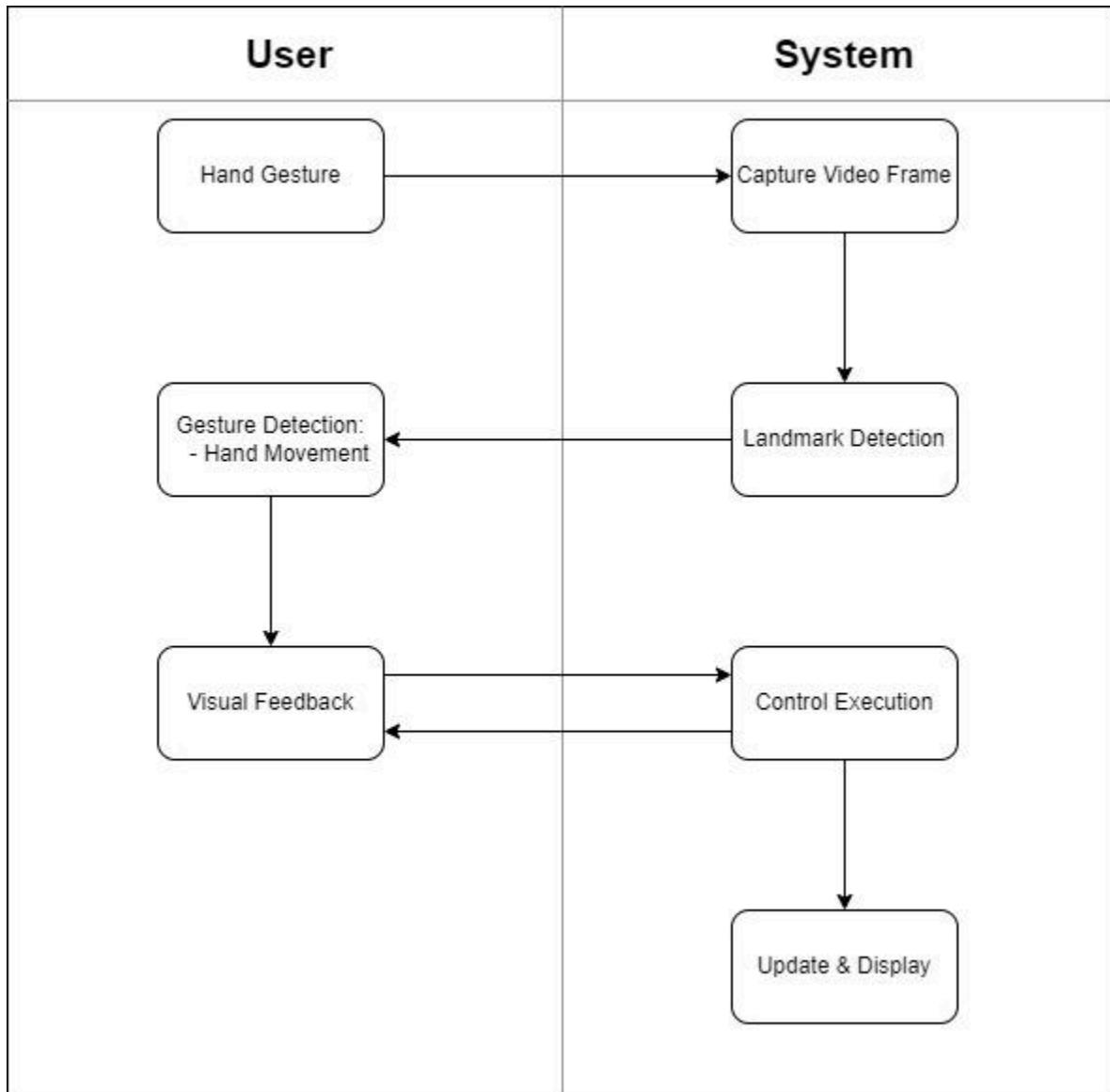


2.2.3 DFD LEVEL 2:





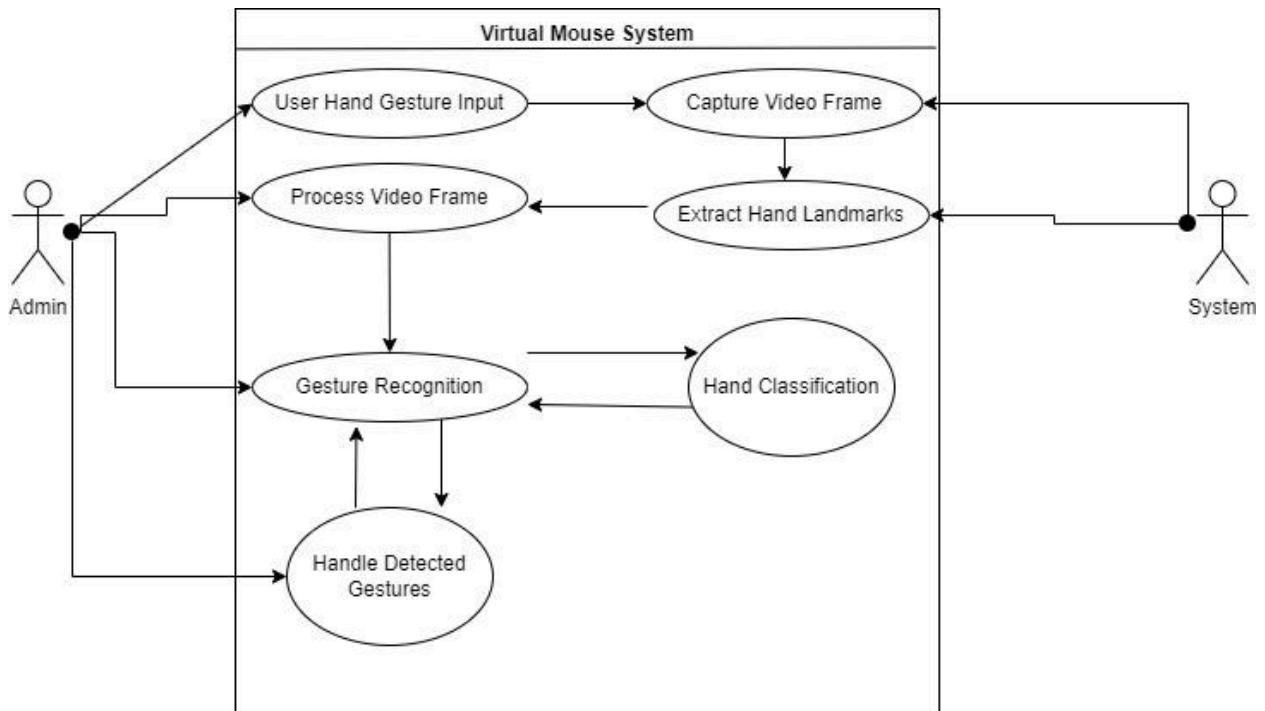
2.3 CALL AND RETURN ARCHITECTURE



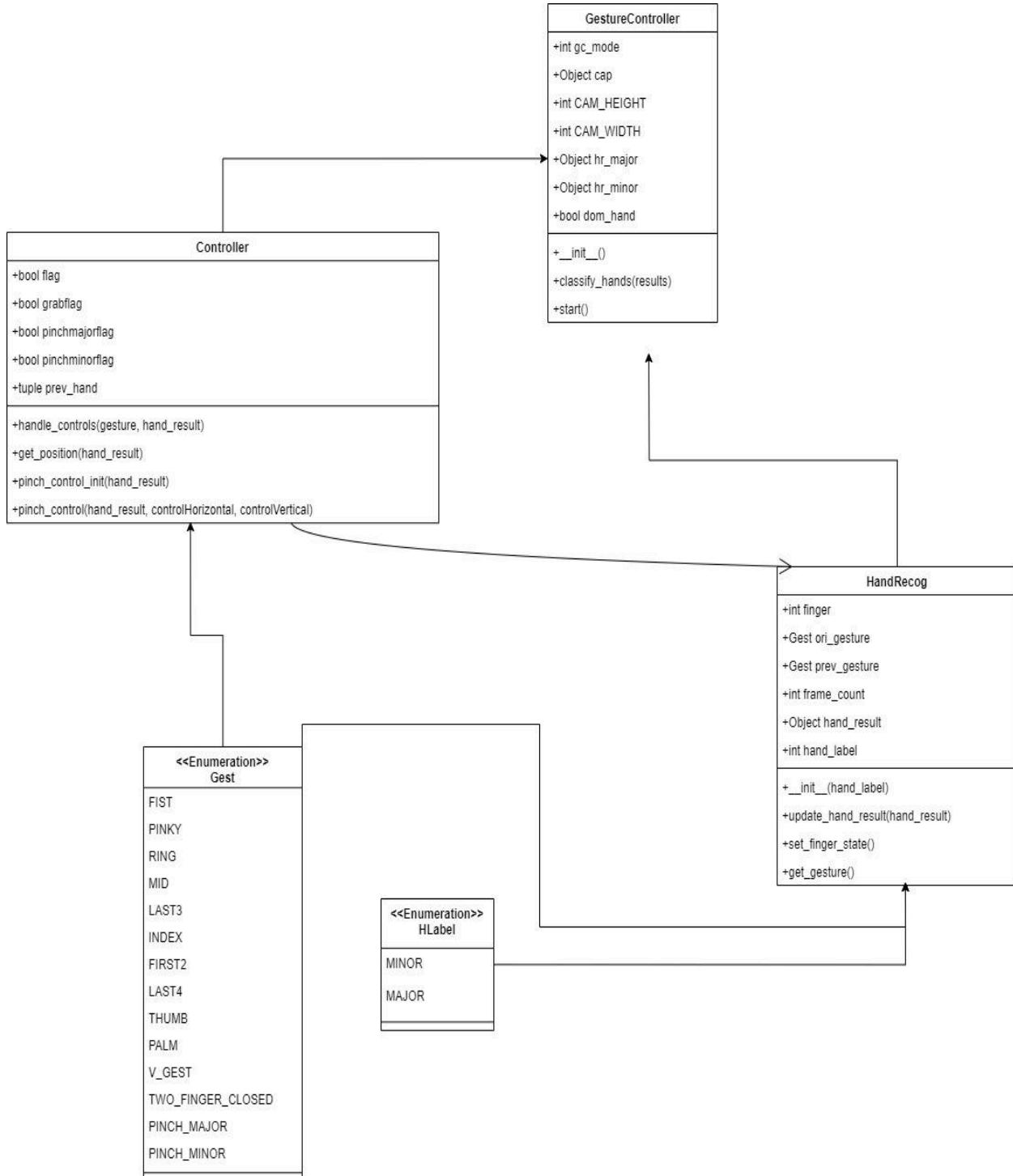


CHAPTER 3: UML DIAGRAM

3.1 USE CASE DIAGRAM

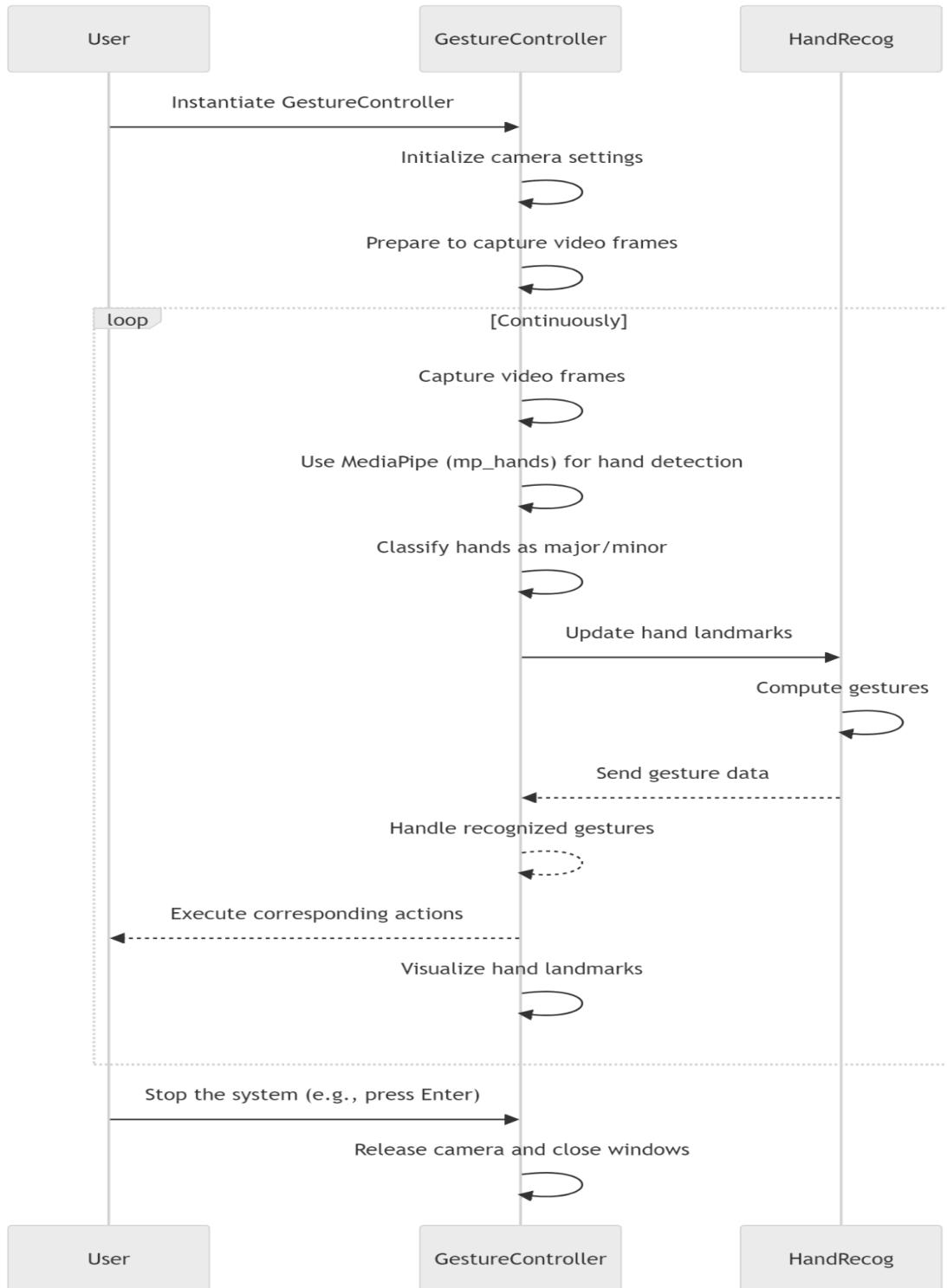


3.2 CLASS DIAGRAM



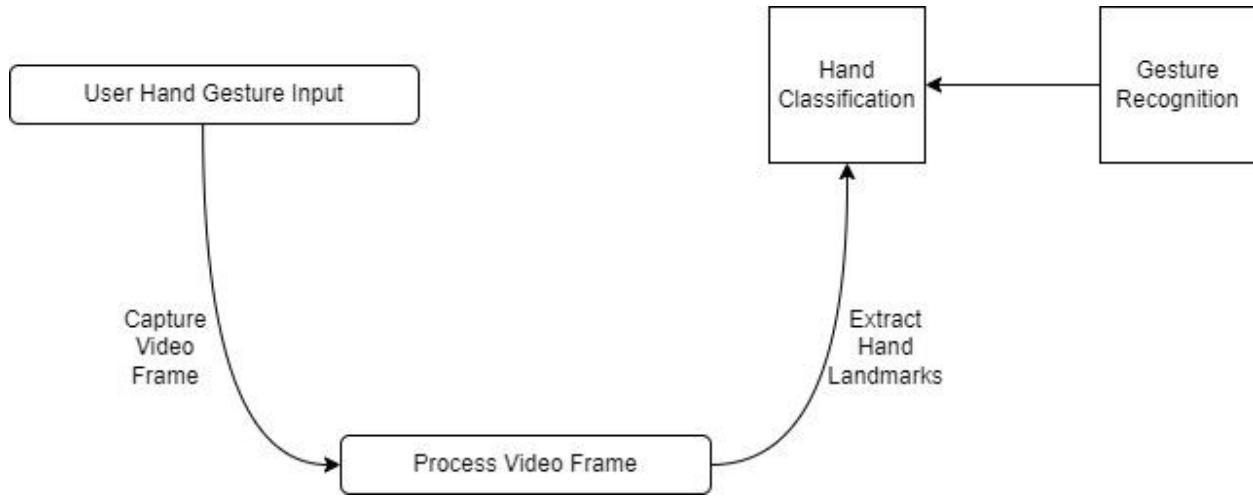


3.3 SEQUENCE DIAGRAM



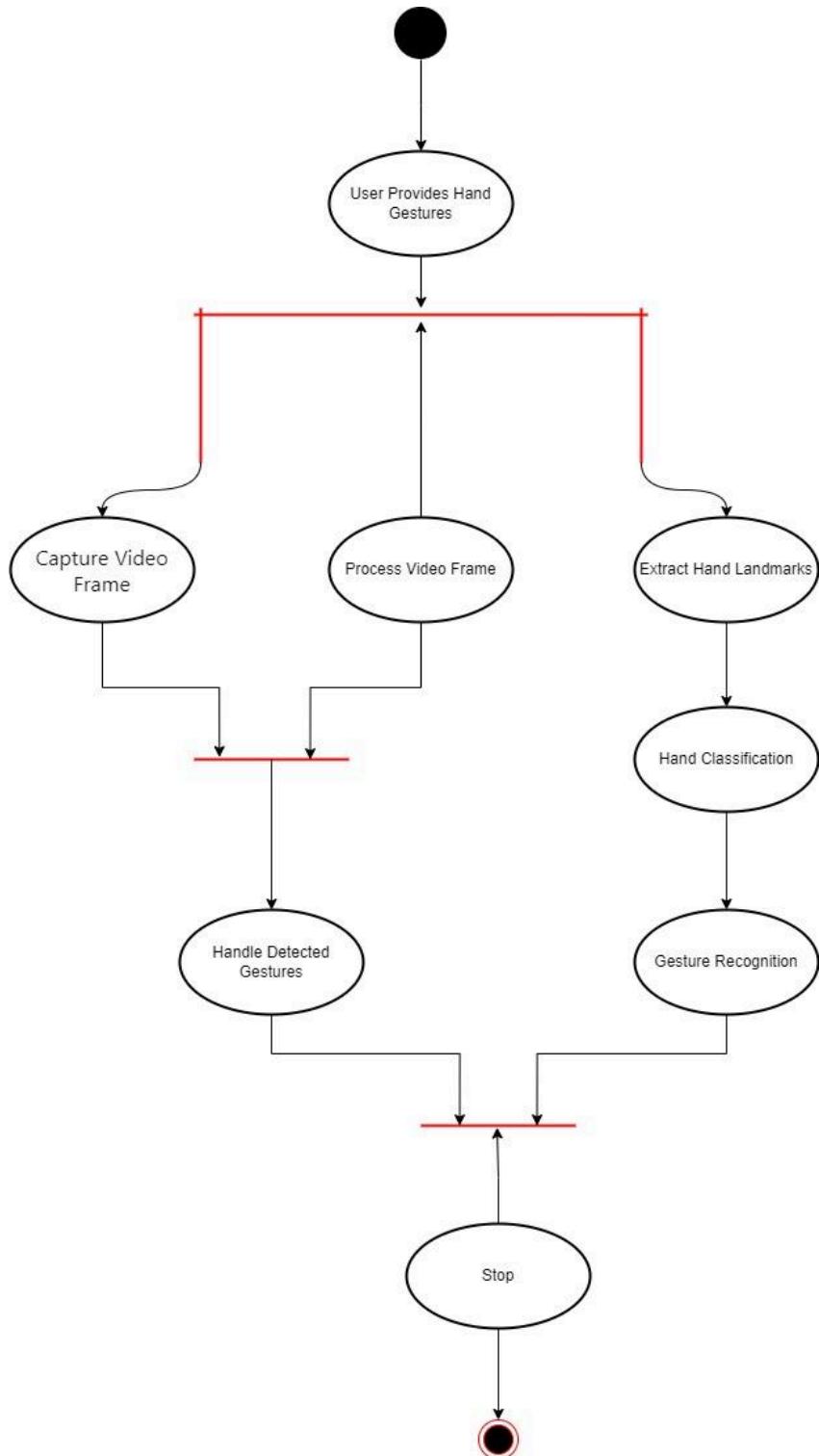


3.4 COLLABORATION DIAGRAM



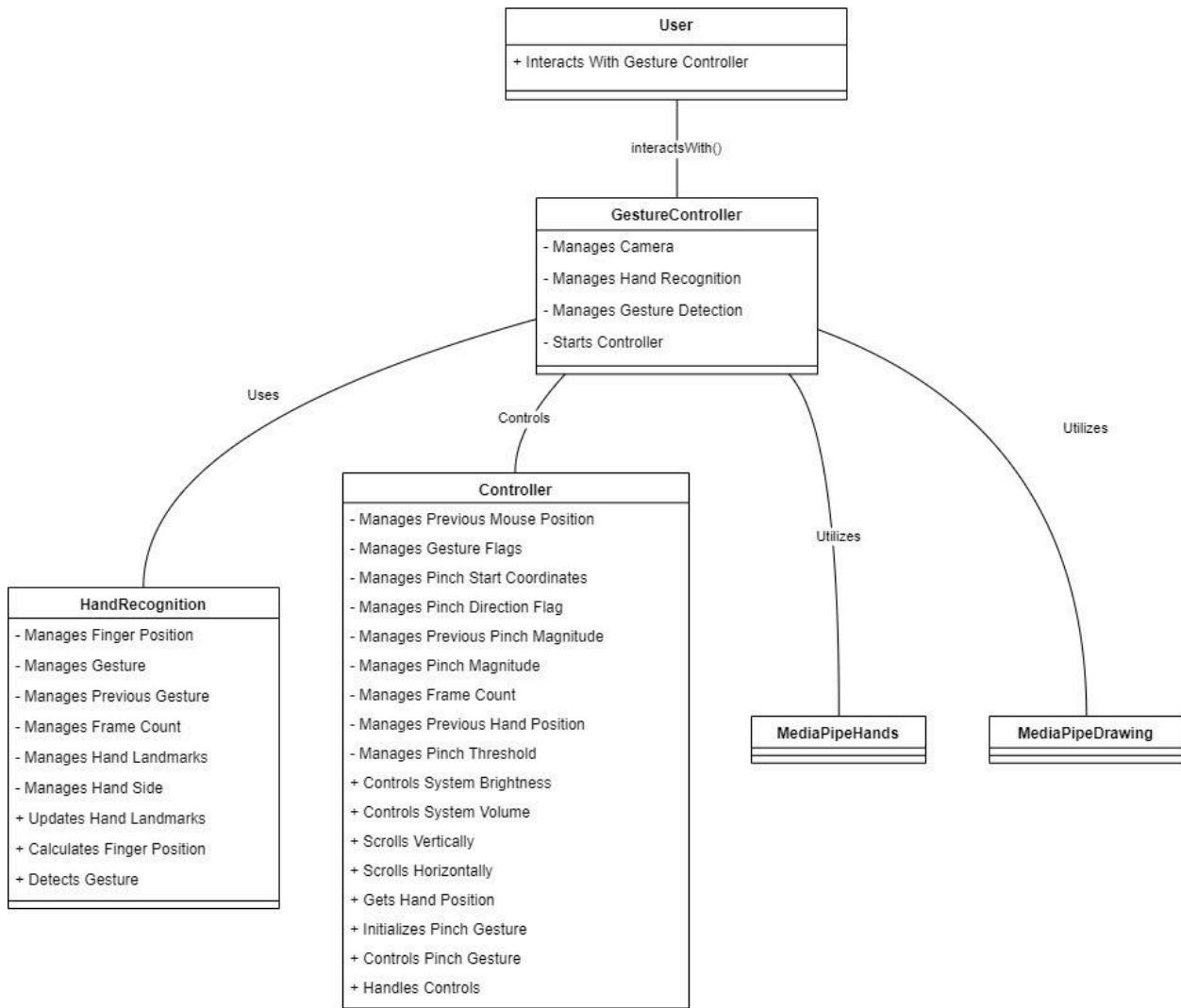


3.5 ACTIVITY DIAGRAM



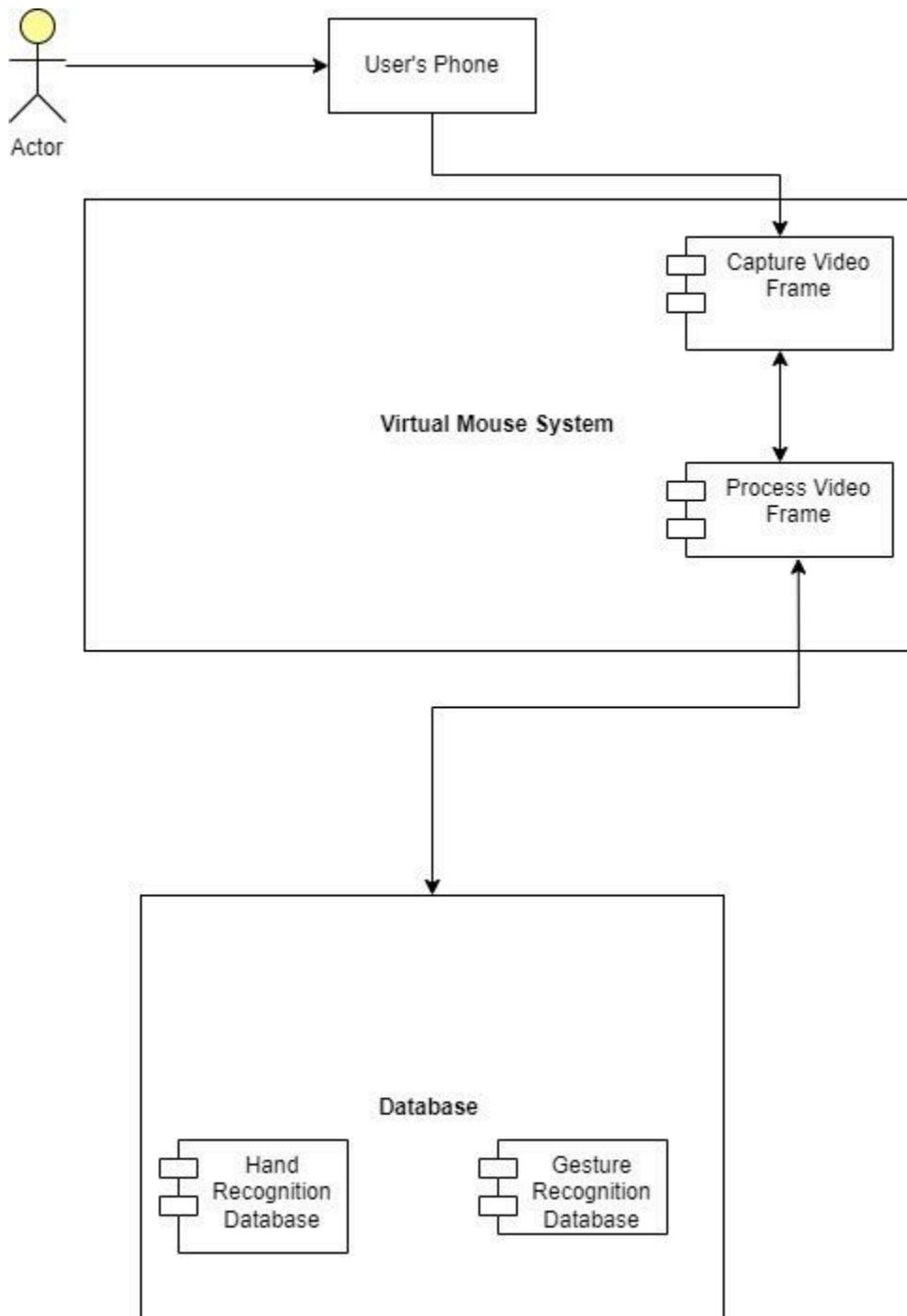


3.6 COMPONENT DIAGRAM





3.7 DEPLOYMENT DIAGRAM



CHAPTER 4: SOFTWARE METRICS

4.1 METHOD TO FIND COST OF SOFTWARE

Software Metrics for Cost and Time Estimation :

Software metrics are quantitative measures that enable the evaluation and improvement of the software development process. By analyzing various aspects of the project, such as size, complexity, and quality, software metrics help in estimating the effort, cost, and time required for development. Here are some software metrics commonly used for cost and time estimation:

1. Size-Oriented Metrics

- **Lines of Code (LOC):** The total number of lines of code in the project, excluding comments and blank lines.
- **Function Points (FP):** An indicator of the overall functionality of the software based on user inputs, outputs, inquiries, files, and interfaces.

2. Effort-Oriented Metrics

- **Person-Months (PM):** The amount of effort required to complete the project, typically measured in person-months.
- **Work Breakdown Structure (WBS):** A hierarchical decomposition of the project into tasks, which helps in estimating effort distribution.

3. Quality-Oriented Metrics

- **Defect Density:** The number of defects found per unit of software size, indicating the quality of the code.

- **Code Coverage:** The percentage of code covered by automated tests, which reflects the thoroughness of testing efforts.

4. Complexity-Oriented Metrics

- **Cyclomatic Complexity:** A measure of the complexity of the software's control flow, indicating the number of independent paths through the code.
- **Halstead Complexity Measures:** Measures of program complexity based on the number of operators and operands used in the code.

4.2 METHOD APPLIED TO FIND THE COST OF SOFTWARE

Method Name: Function Point Analysis

- External Inputs (EI) :

User Input Risk Assessment = 3

User Input Growth Strategy = 3

- External Outputs (EO) :

Generated Risk Assessment Reports = 4

Growth Strategy Recommendations = 4

- External Inquiries (EQ) :

User Queries Risk Assessment = 3

User Queries Growth Strategy = 3

Now let's assign complexity weights:

- Low Complexity Total Weight (L) : 30
- Average Complexity Total Weight (A) : 16
- High Complexity Total Weight (H) : 48

- Calculating Function Points :

The formula provided is:

$$FP = (EI \times L) + (EO \times L) + (EQ \times L) + (ILF \times A) + (EIF \times A)$$

1. External Inputs (EI):

$$EI = 3 + 3 = 6$$

$$EI \times L = 6 \times 30 = 180$$

2. External Outputs (EO):

$$EO = 4 + 4 = 8$$

$$EO \times L = 8 \times 30 = 240$$

3. External Inquiries (EQ):

$$EQ = 3 + 3 = 6$$

$$EQ \times L = 6 \times 30 = 180$$

4. Internal Logical Files (ILF):

We assume ILF count is 0 for this calculation as it's not given.

5. External Interface Files (EIF):

We assume EIF count is 0 for this calculation as it's not given.

$$FP = (6 \times 30) + (8 \times 30) + (6 \times 30) + (0 \times 16) + (0 \times 16)$$

$$FP = 180 + 240 + 180 + 0 + 0$$

$$FP = 600$$



Now, based on historical data or industry benchmarks, we can convert these Function Points into effort hours or cost estimates. Let's assume a productivity factor of 25 hours per Function Point:

$$\text{Effort Hours} = \text{FP} * 5$$

$$\text{Effort Hours} = 600 * 5$$

$$\text{Effort Hours} = 3,000$$

Assuming an hourly rate of \$10 for the development team, we can calculate the cost estimate:

$$\text{Cost} = \text{Effort Hours} * \text{Hourly Rate}$$

$$\text{Cost} = 3,000 * \$10$$

$$\text{Cost} = \$30,000$$

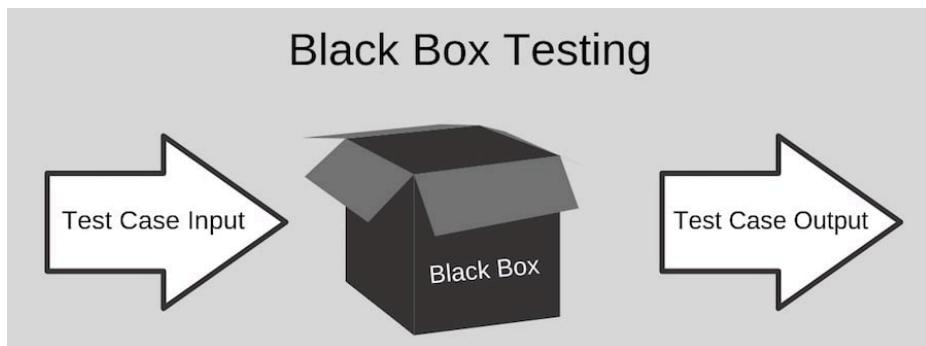
So, based on the estimated effort hours and the assumed hourly rate, the cost estimate for developing the "Real-Time Hand Gesture and Control System using Python" would be approximately \$30,000.



CHAPTER 5: SOFTWARE TESTING

5.1 Black Box testing of any GUI

Black box testing is a testing technique in which the tester does not know how the software works internally. The tester focuses solely on the software's input and output.



- Test cases for black box testing :

Test Case ID	Description	Input	Expected Output	Actual Output	Pass/Fail
TC001	Start application the	Run the script	Camera opens, video feed starts, and GUI window displays the video feed	Camera opened, video feed started, and GUI window displayed the video feed	Pass



TC002	Detect major hand gestures	Show right hand with various gestures	Application correctly identifies gestures (FIST, V_GEST, INDEX, etc.) and executes corresponding actions	Right hand gestures detected correctly and corresponding actions executed	Pass
TC003	Detect minor hand gestures	Show left hand with various gestures	Application correctly identifies gestures (FIST, V_GEST, INDEX, etc.) and executes corresponding actions	Left hand gestures detected correctly and corresponding actions executed	Pass
TC004	Mouse move with V_GEST	Show V_GEST (major hand)	Cursor moves to the hand position	Cursor moved to the hand position	Pass
TC005	Mouse click with MID	Show gesture MID (major hand)	Mouse click at the hand position	Mouse click at the hand position	Pass
TC006	Right-click with INDEX	Show gesture INDEX (major hand)	Mouse right-click at the hand position	Mouse right-click at the hand position	Pass



TC007	Double-click with TWO_FINGER_CLOSED	Show TWO_FINGER_C LOSED gesture (major hand)	Mouse double-click at the hand position	Mouse double-click at the hand position	Pass
TC008	Drag with FIST	Show FIST gesture (major hand)	Mouse drags from the initial hand position to the new hand position	Mouse dragged from the initial hand position to the new hand position	Pass
TC009	Scroll vertically with PINCH_MINOR	Show PINCH_MINOR gesture (minor hand)	Vertical scroll occurs on the screen	Vertical scroll occurred on the screen	Pass
TC010	Scroll horizontally with PINCH_MINOR	Move PINCH_MINOR gesture horizontally	Horizontal scroll occurs on the screen	Horizontal scroll occurred on the screen	Pass
TC011	Close application	Click "X"	Application closes and releases the camera	Application closed and camera released	Pass

1.1 White box testing of any running code.

Unlike black box testing, which focuses solely on software functionality, white box testing methods examine the internal architecture, code structure, employed data structures, and overall software functioning. It is also known as structural testing, glass box testing, or clear box testing. Transparent testing or open box testing are other names for white box testing.



QA testers



Black box - we do not know anything

Developers



White box - we know everything

Code snippet 1:

```
class HandRecog:

    def __init__(self, hand_label):

        self.finger = 0
        self_ori_gesture = Gest.PALM
        self.prev_gesture = Gest.PALM
        self.frame_count = 0
        self.hand_result = None
        self.hand_label = hand_label

    def update_hand_result(self, hand_result):

        self.hand_result = hand_result

    def get_signed_dist(self, point):

        sign = -1
        if self.hand_result.landmark[point[0]].y < self.hand_result.landmark[point[1]].y:
            sign = 1
        dist = (self.hand_result.landmark[point[0]].x - self.hand_result.landmark[point[1]].x) ** 2
        dist += (self.hand_result.landmark[point[0]].y - self.hand_result.landmark[point[1]].y) ** 2
        dist = math.sqrt(dist)
        return dist * sign

    def get_dist(self, point):

        dist = (self.hand_result.landmark[point[0]].x - self.hand_result.landmark[point[1]].x) ** 2
        dist += (self.hand_result.landmark[point[0]].y - self.hand_result.landmark[point[1]].y) ** 2
        dist = math.sqrt(dist)
        return dist
```

Code snippet 2:

```

def get_dz(self, point):

    return abs(self.hand_result.landmark[point[0]].z -
self.hand_result.landmark[point[1]].z)
def set_finger_state(self):

    if self.hand_result == None:
        return

    points = [[8, 5, 0], [12, 9, 0], [16, 13, 0], [20, 17, 0]]
    self.finger = 0
    self.finger = self.finger | 0 # thumb
    for idx, point in enumerate(points):

        dist = self.get_signed_dist(point[:2])
        dist2 = self.get_signed_dist(point[1:])

        try:
            ratio = round(dist / dist2, 1)
        except:
            ratio = round(dist1 / 0.01, 1)

        self.finger = self.finger << 1
        if ratio > 0.5:
            self.finger = self.finger | 1

def get_gesture(self):

    if self.hand_result == None:
        return Gest.PALM

    current_gesture = Gest.PALM
    if self.finger in [Gest.LAST3, Gest.LAST4] and self.get_dist([8, 4]) < 0.05:
        if self.hand_label == HLabel.MINOR:
            current_gesture = Gest.PINCH_MINOR
        else:
            current_gesture = Gest.PINCH_MAJOR

    elif Gest.FIRST2 == self.finger:
        point = [[8, 12], [5, 9]]
        dist1 = self.get_dist(point[0])
        dist2 = self.get_dist(point[1])
        ratio = dist1 / dist2
        if ratio > 1.7:
            current_gesture = Gest.V_GEST
        else:
            if self.get_dz([8, 12]) < 0.1:
                current_gesture = Gest.TWO_FINGER_CLOSED
            else:
                current_gesture = Gest.MID

    else:
        current_gesture = self.finger

    if current_gesture == self.prev_gesture:
        self.frame_count += 1
    else:
        self.frame_count = 0

```

Code snippet 3:

```

class Controller:

    tx_old = 0
    ty_old = 0
    trial = True
    flag = False
    grabflag = False
    pinchmajorflag = False
    pinchminorflag = False
    pinchstartxcoord = None
    pinchstartycoord = None
    pinchdirectionflag = None
    prevpinchlv = 0
    pinchlv = 0
    framecount = 0
    prev_hand = None
    pinch_threshold = 0.3

    def getpinchylv(hand_result):
        dist = round((Controller.pinchstartycoord - hand_result.landmark[8].y) * 10, 1)
        return dist

    def getpinchxlv(hand_result):
        dist = round((hand_result.landmark[8].x - Controller.pinchstartxcoord) * 10, 1)
        return dist

    def changesystembrightness():
        currentBrightnessLv = sbcontrol.get_brightness(display=0) / 100.0
        currentBrightnessLv += Controller.pinchlv / 50.0
        if currentBrightnessLv > 1.0:
            currentBrightnessLv = 1.0
        elif currentBrightnessLv < 0.0:
            currentBrightnessLv = 0.0
        sbcontrol.fade_brightness(int(100 * currentBrightnessLv),
start=sbcontrol.get_brightness(display=0))

    def changesystemvolume():
        devices = AudioUtilities.GetSpeakers()
        interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
        volume = cast(interface, POINTER(IAudioEndpointVolume))
        currentVolumeLv = volume.GetMasterVolumeLevelScalar()
        currentVolumeLv += Controller.pinchlv / 50.0
        if currentVolumeLv > 1.0:
            currentVolumeLv = 1.0
        elif currentVolumeLv < 0.0:
            currentVolumeLv = 0.0
        volume.SetMasterVolumeLevelScalar(currentVolumeLv, None)

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

    def scrollHorizontal():
        pyautogui.keyDown('shift')
        pyautogui.keyDown('ctrl')
        pyautogui.scroll(-120 if Controller.pinchlv > 0.0 else 120)
        pyautogui.keyUp('ctrl')
        pyautogui.keyUp('shift')

    def get_position(hand_result):
        point = 9
        position = [hand_result.landmark[point].x, hand_result.landmark[point].y]
        sx, sy = pyautogui.size()
        x_old, y_old = pyautogui.position()
        x = int(position[0] * sx)
        y = int(position[1] * sy)
        if Controller.prev_hand is None:
            Controller.prev_hand = x, y
        delta_x = x - Controller.prev_hand[0]
        delta_y = y - Controller.prev_hand[1]

        distsq = delta_x ** 2 + delta_y ** 2
        ratio = 1
        Controller.prev_hand = [x, y]

        if distsq <= 25:
            ratio = 0
        elif distsq <= 900:
            ratio = 0.07 * (distsq ** (1 / 2))
        else:
            ratio = 2.1
        x, y = x_old + delta_x * ratio, y_old + delta_y * ratio
        return (x, y)

    def pinch_control_init(hand_result):

```



❖ White box testing on the code snippet for “Real-Time Hand Gesture Recognition and Control System using Python” :

1. Input Validation Testing

- Validate valid input gestures like FIST, V_GEST, INDEX.
- Test multiple gestures in sequence.
- Test invalid input values like undefined gestures or random movements, objects, or non-hand entities.
- Test mixed gestures and transitions.

2. Functionality Testing

- Test individual functions for gesture detection, control functions (mouse move, click, scroll), and application functions (start and close).

3. Data Processing Testing

- Test data processing with various input data like different lighting conditions, backgrounds, and multiple hands with different skin tones.

4. Error Handling Testing

- Test error scenarios like camera access issues, unexpected gestures or positions, and high system load.

5. Integration Testing

- Test camera integration, gesture detection, and system control integration points.

6. Performance Testing

- Test scalability and efficiency with continuous input over time and measure response time for real-time performance.

7. Security Testing

- Test camera access security, data exposure, and proper handling of control commands.

8. Boundary Testing

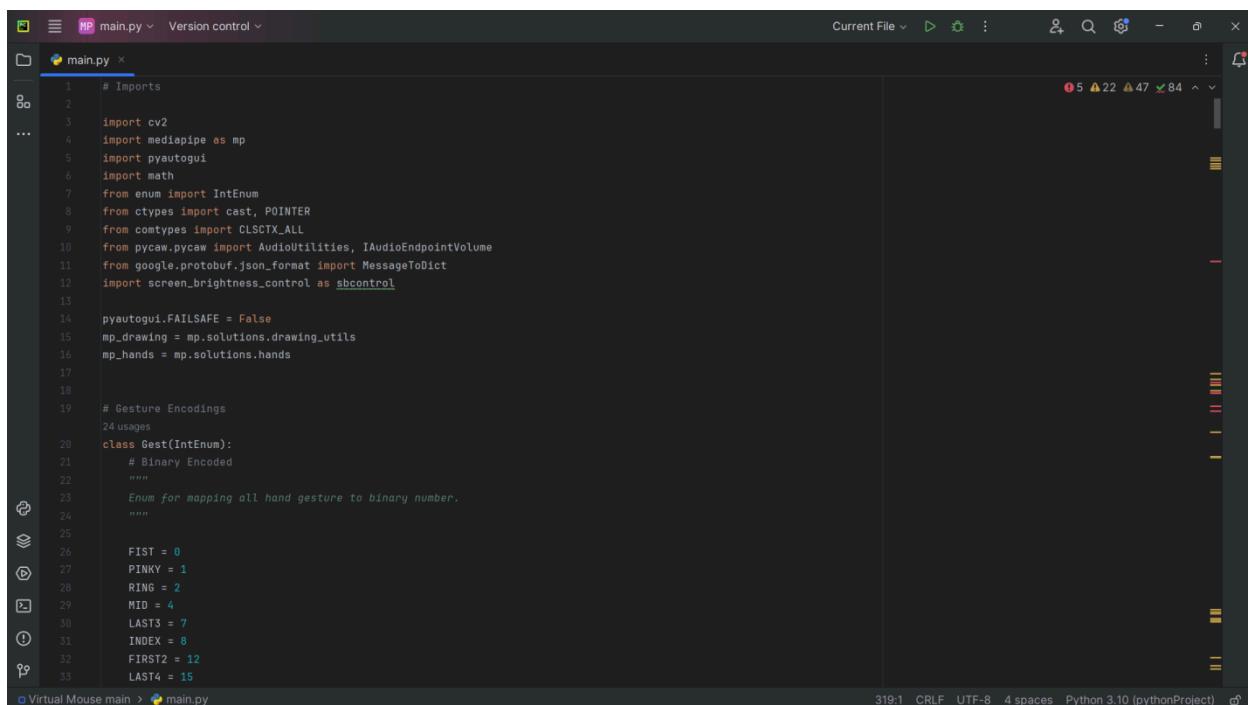
- Test minimal and exaggerated movements and varying hand sizes and distances.

9. Concurrency Testing

- Test multiple simultaneous gestures and multiple user inputs (if applicable).

CHAPTER 6: PROJECT CODE AND OUTPUT

6.1 Running code:



```

1 # Imports
2
3 import cv2
4 import mediapipe as mp
5 import pyautogui
6 import math
7
8 from enum import IntEnum
9 from ctypes import cast, POINTER
10 from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
11 from google.protobuf.json_format import MessageToDict
12 import screen_brightness_control as sbcontrol
13
14 pyautogui.FAILSAFE = False
15 mp_drawing = mp.solutions.drawing_utils
16 mp_hands = mp.solutions.hands
17
18
19 # Gesture Encodings
20 usages
21 class Gest(IntEnum):
22     # Binary Encoded
23     """
24         Enum for mapping all hand gesture to binary number.
25     """
26     FIST = 0
27     PINKY = 1
28     RING = 2
29     MID = 4
30     LAST3 = 7
31     INDEX = 8
32     FIRST2 = 12
33     LAST4 = 15

```



```
main.py
Current File ▾ 319:1 CRLF UTF-8 4 spaces Python 3.10 (pythonProject) ⌂

20 class Gest(IntEnum):
...
25     FIST = 0
26     PINKY = 1
27     RING = 2
28     MID = 4
29     LAST3 = 7
30     INDEX = 8
31     FIRST2 = 12
32     LAST4 = 15
33     THUMB = 16
34     PALM = 31
35
36     # Extra Mappings
37     V_GEST = 33
38     TWO_FINGER_CLOSED = 34
39     PINCH_MAJOR = 35
40     PINCH_MINOR = 36
41
42
43     # Multi-handedness Labels
44     MINOR = 0
45     MAJOR = 1
46
47
48
49
50     # Convert Mediapipe Landmarks to recognizable Gestures
51     MINOR = 0
52     MAJOR = 1
53
54
55     # Convert Mediapipe Landmarks to recognizable Gestures.
56
57     MINOR = 0
58     MAJOR = 1
59
60
61
62     # Convert Mediapipe Landmarks to recognizable Gestures.
63     MINOR = 0
64     MAJOR = 1
65
66
67
68     # Convert Mediapipe Landmarks to recognizable Gestures.
69     MINOR = 0
70     MAJOR = 1
71
72
73     # Convert Mediapipe Landmarks to recognizable Gestures.
74     MINOR = 0
75     MAJOR = 1
76
77
78
79     # Convert Mediapipe Landmarks to recognizable Gestures.
80     MINOR = 0
81     MAJOR = 1
82
83
84
85

Virtual Mouse main > main.py
```

```
main.py
Current File ▾ 319:1 CRLF UTF-8 4 spaces Python 3.10 (pythonProject) ⌂

51     def __init__(self, hand_label):
52         """
53             Constructs all the necessary attributes for the HandRecog object.
54
55             Parameters
56             -----
57             finger : int
58                 Represent gesture corresponding to Enum 'Gest',
59                 stores computed gesture for current frame.
60             ori_gesture : int
61                 Represent gesture corresponding to Enum 'Gest',
62                 stores gesture being used.
63             prev_gesture : int
64                 Represent gesture corresponding to Enum 'Gest',
65                 stores gesture computed for previous frame.
66             frame_count : int
67                 total no. of frames since 'ori_gesture' is updated.
68             hand_result : Object
69                 Landmarks obtained from mediapipe.
70             hand_label : int
71                 Represents multi-handedness corresponding to Enum 'HLabel'.
72
73         """
74
75         self.finger = 0
76         self.ori_gesture = Gest.PALM
77         self.prev_gesture = Gest.PALM
78         self.frame_count = 0
79         self.hand_result = None
80         self.hand_label = hand_label
81
82
83
84
85

Virtual Mouse main > main.py
```



main.py

```
51  class HandRecog:
52      2 usages
53
54      def update_hand_result(self, hand_result):
55          self.hand_result = hand_result
56
57
58      2 usages
59      def get_signed_dist(self, point):
60          """
61              returns signed euclidean distance between 'point'.
62
63              Parameters
64              -----
65              point : list containing two elements of type list/tuple which represents
66                  | landmark point.
67
68              Returns
69              -----
70              float
71              """
72              sign = -1
73              if self.hand_result.landmark[point[0]].y < self.hand_result.landmark[point[1]].y:
74                  sign = 1
75              dist = (self.hand_result.landmark[point[0]].x - self.hand_result.landmark[point[1]].x) ** 2
76              dist += (self.hand_result.landmark[point[0]].y - self.hand_result.landmark[point[1]].y) ** 2
77              dist = math.sqrt(dist)
78              return dist * sign
79
80
81      3 usages
82      def get_dist(self, point):
83          """
84              returns euclidean distance between 'point'.
85
86              Parameters
87              -----
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
```

Virtual Mouse main > main.py

main.py

```
51  class HandRecog:
52      2 usages
53
54      def update_hand_result(self, hand_result):
55          self.hand_result = hand_result
56
57
58      2 usages
59      def get_dist(self, point):
60          """
61              Parameters
62              -----
63              point : list containing two elements of type list/tuple which represents
64                  | landmark point.
65
66              Returns
67              -----
68              float
69              """
70              dist = (self.hand_result.landmark[point[0]].x - self.hand_result.landmark[point[1]].x) ** 2
71              dist += (self.hand_result.landmark[point[0]].y - self.hand_result.landmark[point[1]].y) ** 2
72              dist = math.sqrt(dist)
73              return dist
74
75
76      1 usage
77      def get_dz(self, point):
78          """
79              returns absolute difference on z-axis between 'point'.
80
81              Parameters
82              -----
83              point : list containing two elements of type list/tuple which represents
84                  | landmark point.
85
86              Returns
87              -----
88              float
89              """
90              return abs(self.hand_result.landmark[point[0]].z - self.hand_result.landmark[point[1]].z)
91
92
93      # Function to find Gesture Encoding using current finger_state.
94      # Finger_state: 1 if finger is open, else 0
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
```

Virtual Mouse main > main.py



AISSMS

INSTITUTE OF INFORMATION TECHNOLOGY
ADDING VALUE TO ENGINEERING

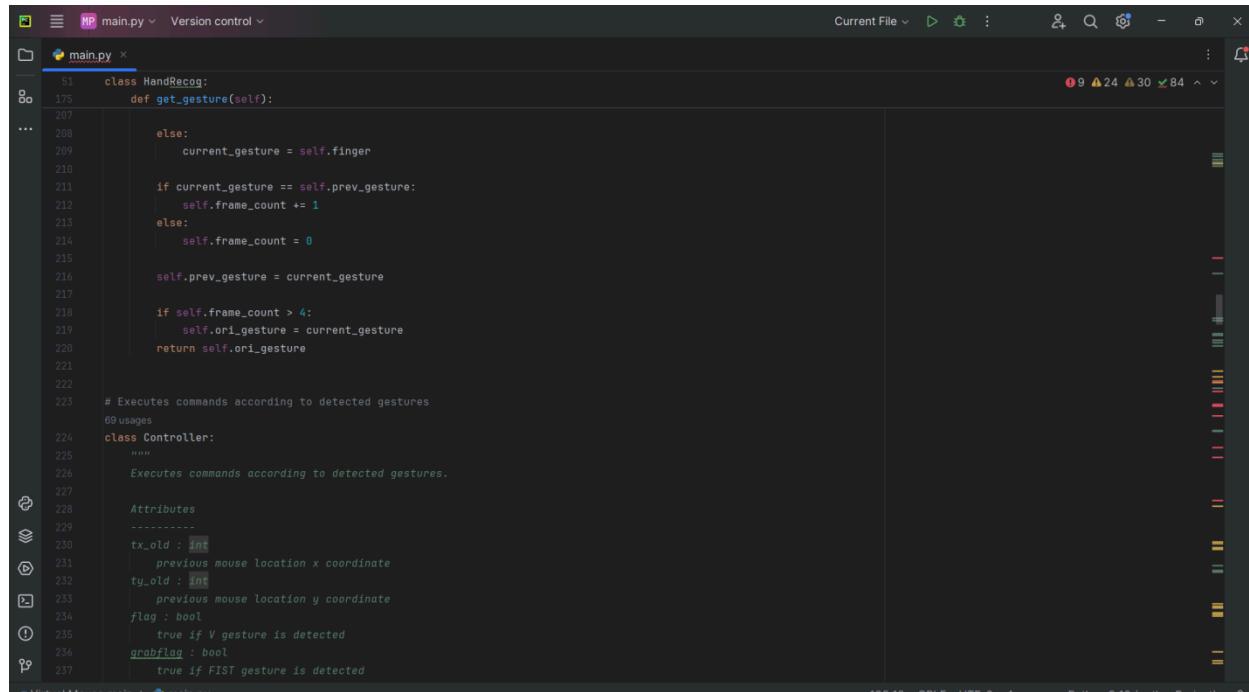


```
main.py  Version control  Current File  D  S  Q  E  X  0 5 22 46 84 ^ v

51 class HandRecog:
52     2 usages
53     def set_finger_state(self):
54         """
55             set 'finger' by computing ratio of distance between finger tip
56             , middle knuckle, base knuckle.
57
58             Returns
59             -----
60             None
61             """
62             if self.hand_result == None:
63                 return
64
65             points = [[8, 5, 0], [12, 9, 0], [16, 13, 0], [20, 17, 0]]
66             self.finger = 0
67             self.finger = self.finger | 0 # thumb
68             for idx, point in enumerate(points):
69
70                 dist = self.get_signed_dist(point[:2])
71                 dist2 = self.get_signed_dist(point[1:])
72
73             try:
74                 ratio = round(dist / dist2, 1)
75             except:
76                 ratio = round(dist1 / 0.01, 1)
77
78             self.finger = self.finger << 1
79             if ratio > 0.5:
80                 self.finger = self.finger | 1
81
82             # Handling Fluctuations due to noise
83             2 usages
84             def get_gesture(self):
85
86                 # Virtual Mouse main > main.py  319:1 CRLF UTF-8 4 spaces Python 3.10 (pythonProject) 0
```

```
main.py  Version control  Current File  D  S  Q  E  X  0 9 24 30 84 ^ v

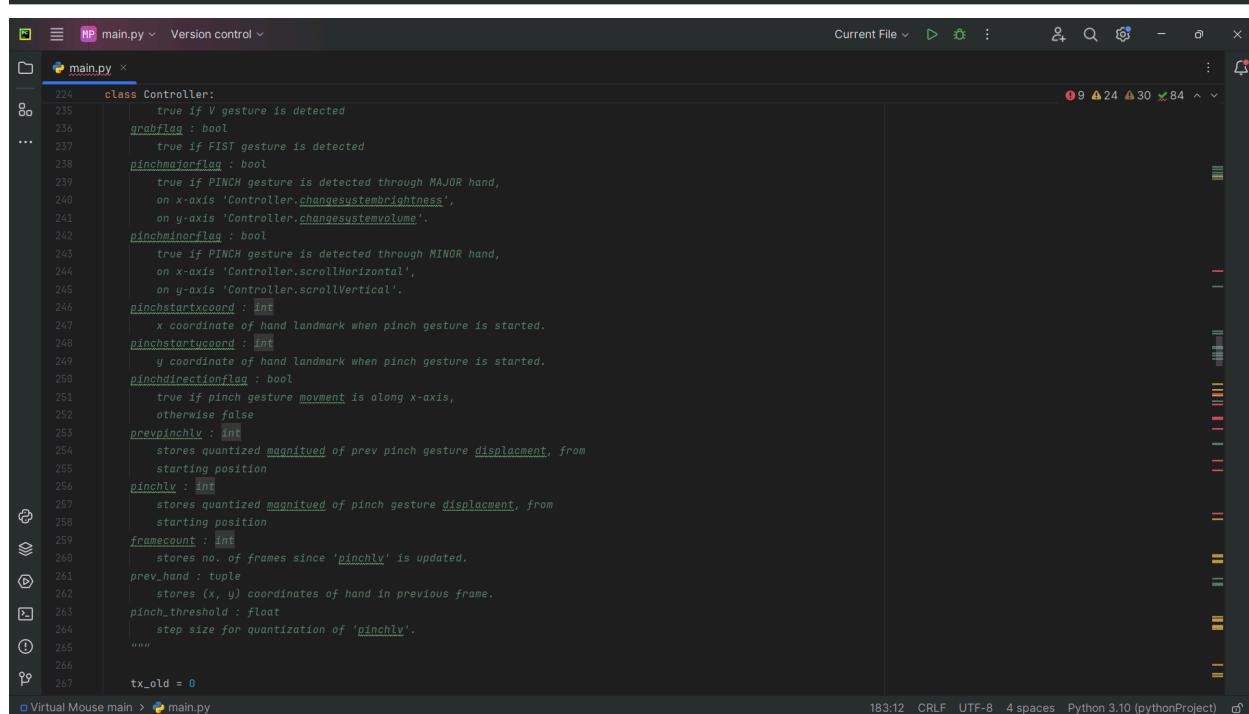
51 class HandRecog:
52     # Handling Fluctuations due to noise
53     2 usages
54     def get_gesture(self):
55         """
56             returns int representing gesture corresponding to Enum 'Gest'.
57             sets 'frame_count', 'ori_gesture', 'prev_gesture',
58             handles fluctuations due to noise.
59
60             Returns
61             -----
62             int
63             """
64             if self.hand_result == None:
65                 return Gest.PALM
66
67             current_gesture = Gest.PALM
68             if self.finger in [Gest.LAST3, Gest.LAST4] and self.get_dist([8, 4]) < 0.05:
69                 if self.hand_label == HLabel.MINOR:
70                     current_gesture = Gest.PINCH_MINOR
71                 else:
72                     current_gesture = Gest.PINCH_MAJOR
73
74             elif Gest.FIRST2 == self.finger:
75                 point = [[8, 12], [5, 9]]
76                 dist1 = self.get_dist(point[0])
77                 dist2 = self.get_dist(point[1])
78                 ratio = dist1 / dist2
79                 if ratio > 1.7:
80                     current_gesture = Gest.V_GEST
81                 else:
82                     if self.get_dz([8, 12]) < 0.1:
83                         current_gesture = Gest.TWO_FINGER_CLOSED
84                     else:
```



```

51     class HandRecog:
52         def get_gesture(self):
53             ...
54             else:
55                 current_gesture = self.finger
56
57             if current_gesture == self.prev_gesture:
58                 self.frame_count += 1
59             else:
60                 self.frame_count = 0
61
62             self.prev_gesture = current_gesture
63
64             if self.frame_count > 4:
65                 self.ori_gesture = current_gesture
66             return self.ori_gesture
67
68
69 # Executes commands according to detected gestures
70 usages
71
72 class Controller:
73     """
74     Executes commands according to detected gestures.
75
76     Attributes
77     -----
78     tx_old : int
79         previous mouse location x coordinate
80     ty_old : int
81         previous mouse location y coordinate
82     flag : bool
83         true if V gesture is detected
84     grabflag : bool
85         true if FIST gesture is detected
86
87     """
88
89     tx_old = 0

```



```

224     class Controller:
225         """
226             true if V gesture is detected
227         grabflag : bool
228             true if FIST gesture is detected
229         pinchmajorflag : bool
230             true if PINCH gesture is detected through MAJOR hand,
231             on x-axis 'Controller.changesystembrightness',
232             on y-axis 'Controller.changesystemvolume'.
233         pinchminorflag : bool
234             true if PINCH gesture is detected through MINOR hand,
235             on x-axis 'Controller.scrollHorizontal',
236             on y-axis 'Controller.scrollVertical'.
237         pinchstartxcoord : int
238             x coordinate of hand landmark when pinch gesture is started.
239         pinchstartycoord : int
240             y coordinate of hand landmark when pinch gesture is started.
241         pinchdirectionflag : bool
242             true if pinch gesture movement is along x-axis,
243             otherwise false
244         prevpinchly : int
245             stores quantized magnituded of prev pinch gesture displacement, from
246             starting position
247         pinchly : int
248             stores quantized magnituded of pinch gesture displacement, from
249             starting position
250         framecount : int
251             stores no. of frames since 'pinchly' is updated.
252         prev_hand : tuple
253             stores (x, y) coordinates of hand in previous frame.
254         pinch_threshold : float
255             step size for quantization of 'pinchly'.
256
257     """
258
259     tx_old = 0

```



main.py

```

224     class Controller:
267         tx_old = 0
268         ty_old = 0
269         trial = True
270         flag = False
271         grabFlag = False
272         pinchmajorflag = False
273         pinchminorflag = False
274         pinchstartxcoord = None
275         pinchstartycoord = None
276         pinchdirectionflag = None
277         prevpinchlv = 0
278         pinchlv = 0
279         framecount = 0
280         prev_hand = None
281         pinch_threshold = 0.3
282
283     usage
284     def getpinchylv(hand_result):
285         """returns distance between starting pinch y coord and current hand position y coord."""
286         dist = round((Controller.pinchstartycoord - hand_result.landmark[8].y) * 10, 1)
287         return dist
288
289     usage
290     def getpinchxlv(hand_result):
291         """returns distance between starting pinch x coord and current hand position x coord."""
292         dist = round((hand_result.landmark[8].x - Controller.pinchstartxcoord) * 10, 1)
293         return dist
294
295     usage
296     def changesystembrightness():
297         """sets system brightness based on 'Controller.pinchlv'."""
298         currentBrightnessLv = sbcontrol.get_brightness(display=0) / 100.0
299         currentBrightnessLv += Controller.pinchlv / 50.0
300
301     usage
302     def changesystemvolume():
303         """sets system volume based on 'Controller.pinchlv'."""
304         devices = AudioUtilities.GetSpeakers()
305         interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
306         volume = cast(interface, POINTER(IAudioEndpointVolume))
307         currentVolumeLv = volume.GetMasterVolumeLevelScalar()
308         currentVolumeLv += Controller.pinchlv / 50.0
309         if currentVolumeLv > 1.0:
310             currentVolumeLv = 1.0
311         elif currentVolumeLv < 0.0:
312             currentVolumeLv = 0.0
313         volume.SetMasterVolumeLevelScalar(currentVolumeLv, None)
314
315     usage
316     def scrollVertical():
317         """scrolls on screen vertically."""
318         pyautogui.scroll(120 if Controller.pinchlv > 0.0 else -120)
319     def scrollHorizontal():
320         """scrolls on screen horizontally."""
321         pyautogui.keyDown('shift')
322         pyautogui.keyDown('ctrl')
323         pyautogui.scroll(-120 if Controller.pinchlv > 0.0 else 120)
324         pyautogui.keyUp('ctrl')

```

Virtual Mouse main > main.py

main.py

```

224     class Controller:
293     def changesystembrightness():
294         currentBrightnessLv = sbcontrol.get_brightness(display=0) / 100.0
295         currentBrightnessLv += Controller.pinchlv / 50.0
296         if currentBrightnessLv > 1.0:
297             currentBrightnessLv = 1.0
298         elif currentBrightnessLv < 0.0:
299             currentBrightnessLv = 0.0
300         sbcontrol.fade_brightness(int(100 * currentBrightnessLv), start=sbcontrol.get_brightness(display=0))
301
302     usage
303     def changesystemvolume():
304         """sets system volume based on 'Controller.pinchlv'."""
305         devices = AudioUtilities.GetSpeakers()
306         interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
307         volume = cast(interface, POINTER(IAudioEndpointVolume))
308         currentVolumeLv = volume.GetMasterVolumeLevelScalar()
309         currentVolumeLv += Controller.pinchlv / 50.0
310         if currentVolumeLv > 1.0:
311             currentVolumeLv = 1.0
312         elif currentVolumeLv < 0.0:
313             currentVolumeLv = 0.0
314         volume.SetMasterVolumeLevelScalar(currentVolumeLv, None)
315
316     usage
317     def scrollVertical():
318         """scrolls on screen vertically."""
319         pyautogui.scroll(120 if Controller.pinchlv > 0.0 else -120)
320     def scrollHorizontal():
321         """scrolls on screen horizontally."""
322         pyautogui.keyDown('shift')
323         pyautogui.keyDown('ctrl')
324         pyautogui.scroll(-120 if Controller.pinchlv > 0.0 else 120)
325         pyautogui.keyUp('ctrl')

```

Virtual Mouse main > main.py



```
327     # Stabilize cursor by Dampening_
328     def get_position(hand_result):
329         """
330         returns coordinates of current hand position.
331 
332         Locates hand to get cursor position also stabilize cursor by
333         dampening jerky motion of hand.
334 
335         Returns
336         -----
337         tuple(float, float)
338         """
339         point = 9
340         position = [hand_result.landmark[point].x, hand_result.landmark[point].y]
341         sx, sy = pautogui.size()
342         x_old, y_old = pautogui.position()
343         x = int(position[0] * sx)
344         y = int(position[1] * sy)
345         if Controller.prev_hand is None:
346             Controller.prev_hand = x, y
347             delta_x = x - Controller.prev_hand[0]
348             delta_y = y - Controller.prev_hand[1]
349 
350             distsq = delta_x ** 2 + delta_y ** 2
351             ratio = 1
352             Controller.prev_hand = [x, y]
353 
354             if distsq <= 25:
355                 ratio = 0
356             elif distsq <= 900:
357                 ratio = 0.07 * (distsq ** (1 / 2))
358             else:
359                 ratio = 2.1
360             x, y = x_old + delta_x * ratio, y_old + delta_y * ratio
361 
362     Virtual Mouse main > main.py
```

```
363     def pinch_control_init(hand_result):
364         """Initializes attributes for pinch gesture."""
365         Controller.pinchstartxcoord = hand_result.landmark[8].x
366         Controller.pinchstartycoord = hand_result.landmark[8].y
367         Controller.pinchlv = 0
368         Controller.prevpinchlv = 0
369         Controller.framecount = 0
370 
371         # Hold final position for 5 frames to change status_
372         def pinch_control(hand_result, controlHorizontal, controlVertical):
373             """
374                 calls 'controlHorizontal' or 'controlVertical' based on pinch flags,
375                 'framecount' and sets 'pinchlv'.
376 
377             Parameters
378             -----
379             hand_result : Object
380                 Landmarks obtained from mediapipe.
381             controlHorizontal : callback function associated with horizontal
382                 pinch gesture.
383             controlVertical : callback function associated with vertical
384                 pinch gesture.
385 
386             Returns
387             -----
388             None
389             """
390             if Controller.framecount == 5:
391                 Controller.framecount = 0
392                 Controller.pinchlv = Controller.prevpinchlv
393 
394             if Controller.pinchdirectionflag == True:
395                 controlHorizontal() # x
396 
397     Virtual Mouse main > main.py
```



```

main.py  Version control  Current File  19  24  30  84  ▾
main.py  Version control  Current File  19  24  30  84  ▾

```

main.py

```

372     def pinch_control(hand_result, controlHorizontal, controlVertical):
394         if Controller.pinchdirectionflag == True:
395             controlHorizontal() # x
396
397         elif Controller.pinchdirectionflag == False:
398             controlVertical() # y
399
400         lvx = Controller.getpinchxlv(hand_result)
401         lvy = Controller.getpinchylv(hand_result)
402
403         if abs(lvy) > abs(lvx) and abs(lvy) > Controller.pinch_threshold:
404             Controller.pinchdirectionflag = False
405             if abs(Controller.prevpinchlv - lvy) < Controller.pinch_threshold:
406                 Controller.framecount += 1
407             else:
408                 Controller.prevpinchlv = lvy
409                 Controller.framecount = 0
410
411             elif abs(lvx) > Controller.pinch_threshold:
412                 Controller.pinchdirectionflag = True
413                 if abs(Controller.prevpinchlv - lvx) < Controller.pinch_threshold:
414                     Controller.framecount += 1
415                 else:
416                     Controller.prevpinchlv = lvx
417                     Controller.framecount = 0
418
419     def handle_controls(gesture, hand_result):
420         """Implements all gesture functionality."""
421         x, y = None, None
422         if gesture != Gest.PALM:
423             x, y = Controller.get_position(hand_result)
424
425         # flag reset
426         if gesture != Gest.FIST and Controller.grabflag:
427             pyautogui.mouseUp(button="left")
428
429         if gesture != Gest.PINCH_MAJOR and Controller.pinchmajorflag:
430             Controller.pinchmajorflag = False
431
432         if gesture != Gest.PINCH_MINOR and Controller.pinchminorflag:
433             Controller.pinchminorflag = False
434
435         # implementation
436         if gesture == Gest.V_GEST:
437             Controller.flag = True
438             pyautogui.moveTo(x, y, duration=0.1)
439
440         elif gesture == Gest.FIST:
441             if not Controller.grabflag:
442                 Controller.grabflag = True
443                 pyautogui.mouseDown(button="left")
444                 pyautogui.moveTo(x, y, duration=0.1)
445
446         elif gesture == Gest.MID and Controller.flag:
447             pyautogui.click()
448             Controller.flag = False
449
450         elif gesture == Gest.INDEX and Controller.flag:
451             pyautogui.click(button='right')
452             Controller.flag = False
453

```

Virtual Mouse main > main.py

183:12 CRLF UTF-8 4 spaces Python 3.10 (pythonProject)



main.py

```
419     def handle_controls(gesture, hand_result):
420         .....
421         if gesture == Gest.SINGLE_TAP:
422             pyautogui.click()
423             Controller.flag = False
424
425         elif gesture == Gest.INDEX and Controller.flag:
426             pyautogui.click(button='right')
427             Controller.flag = False
428
429         elif gesture == Gest.TWO_FINGER_CLOSED and Controller.flag:
430             pyautogui.doubleClick()
431             Controller.flag = False
432
433         elif gesture == Gest.PINCH_MINOR:
434             if Controller.pinchminorflag == False:
435                 Controller.pinch_control_init(hand_result)
436                 Controller.pinchminorflag = True
437             Controller.pinch_control(hand_result, Controller.scrollHorizontal, Controller.scrollVertical)
438
439         elif gesture == Gest.PINCH_MAJOR:
440             if Controller.pinchmajorflag == False:
441                 Controller.pinch_control_init(hand_result)
442                 Controller.pinchmajorflag = True
443             Controller.pinch_control(hand_result, Controller.changesystembrightness, Controller.changesystemvolume)
444
445         .....
446         Main Class
447         .....
448         Entry point of Gesture Controller
449         .....
450
451     19 usages
452     class GestureController:
453         .....
454
455     Virtual Mouse main > main.py
```

main.py

```
477
478     19 usages
479     class GestureController:
480         """
481             Handles camera, obtain landmarks from mediapipe, entry point
482             for whole program.
483
484             Attributes
485             -----
486             gc_mode : int
487                 indicates weather gesture controller is running or not,
488                 1 if running, otherwise 0.
489             cap : Object
490                 object obtained from cv2, for capturing video frame.
491             CAM_HEIGHT : int
492                 height in pixels of obtained frame from camera.
493             CAM_WIDTH : int
494                 width in pixels of obtained frame from camera.
495             hr_major : Object of 'HandBeog'
496                 object representing major hand.
497             hr_minor : Object of 'HandBeog'
498                 object representing minor hand.
499             dom_hand : bool
500                 True if right hand is dominant hand, otherwise False.
501                 default True.
502
503             gc_mode = 0
504             cap = None
505             CAM_HEIGHT = None
506             CAM_WIDTH = None
507             hr_major = None # Right Hand by default
508             hr_minor = None # Left hand by default
509             dom_hand = True
510
511     Virtual Mouse main > main.py
```



AISSMS

INSTITUTE OF INFORMATION TECHNOLOGY

ADDING VALUE TO ENGINEERING



The screenshot shows a Microsoft Visual Studio Code window with the following details:

- Title Bar:** Shows the project name "HP" and the file "main.py".
- File Explorer:** On the left, it shows a folder structure with "main.py" selected.
- Code Editor:** The main area contains Python code for a "GestureController" class. The code includes methods for initialization, classifying hands, and setting hand modes. It uses OpenCV's VideoCapture and CAP_PROP_FRAME_HEIGHT/WIDTH properties.
- Status Bar:** At the bottom, it displays "509:1 CRLF UTF-8 4 spaces Python 3.10 (pythonProject)".
- Activity Bar:** On the right, there are various status icons including battery level (9%), signal strength (24%), and network (30%).

```
478     class GestureController:
479         """
480             ...
481         gc_mode = 0
482         cap = None
483         CAM_HEIGHT = None
484         CAM_WIDTH = None
485         hr_major = None # Right Hand by default
486         hr_minor = None # Left hand by default
487         dom_hand = True
488
489     def __init__(self):
490         """Initializes attributes."""
491         GestureController.gc_mode = 1
492         GestureController.cap = cv2.VideoCapture(0)
493         GestureController.CAM_HEIGHT = GestureController.cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
494         GestureController.CAM_WIDTH = GestureController.cap.get(cv2.CAP_PROP_FRAME_WIDTH)
495
496     usage
497     def classify_hands(results):
498         """
499             sets 'hr_major', 'hr_minor' based on classification(left, right) of
500             hand obtained from mediapipe, uses 'dom_hand' to decide major and
501             minor hand.
502             """
503             left, right = None, None
504             try:
505                 handedness_dict = MessageToDict(results.multi_handedness[0])
506                 if handedness_dict['classification'][0]['label'] == 'Right':
507                     right = results.multi_hand_landmarks[0]
508                 else:
509                     left = results.multi_hand_landmarks[0]
510             except:
511                 pass
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
```

```
478     class GestureController:
479         def __init__(self):
480             pass
481
482         def classify_hands(self, results):
483             try:
484                 handedness_dict = MessageToDict(results.multi_handedness[1])
485                 if handedness_dict['classification'][0]['label'] == 'Right':
486                     right = results.multi_hand_landmarks[0]
487                 else:
488                     left = results.multi_hand_landmarks[1]
489             except:
490                 pass
491
492             if GestureController.dom_hand == True:
493                 GestureController.hr_major = right
494                 GestureController.hr_minor = left
495             else:
496                 GestureController.hr_major = left
497                 GestureController.hr_minor = right
498
499             1usage
500         def start(self):
501             """
502                 Entry point of whole program, captures video frame and passes, obtains
503                 landmark from mediapipe and passes it to 'handmajor' and 'handminor' for
504                 controlling.
505             """
506
507             handmajor = HandRecog(HLabel.MAJOR)
508             handminor = HandRecog(HLabel.MINOR)
509
510             with mp_hands.Hands(max_num_hands=2, min_detection_confidence=0.5, min_tracking_confidence=0.5) as hands:
511                 while GestureController.cap.isOpened() and GestureController.gc_mode:
512                     success, image = GestureController.cap.read()
```



```

main.py  Version control  Current File  9  24  30  84  ▾
main.py  Version control  Current File  9  24  30  84  ▾

```

Virtual Mouse main > main.py

```

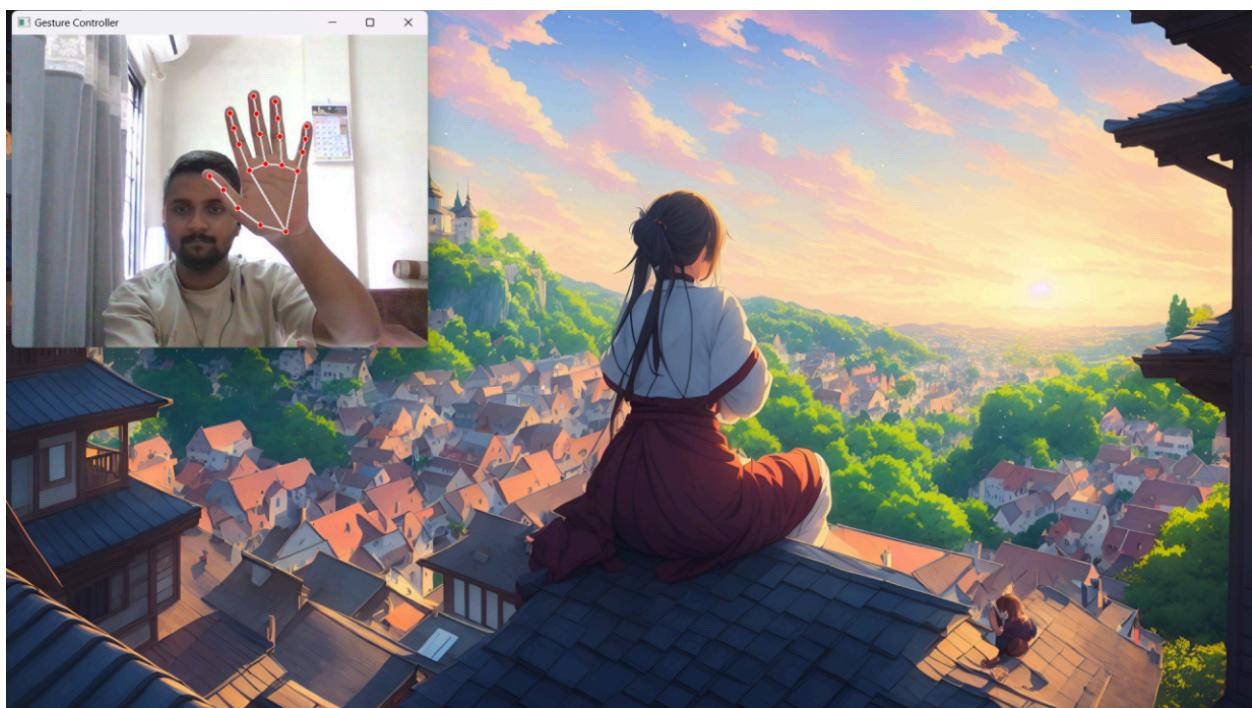
478     class GestureController:
49     def start(self):
50         handmajor = HandRecog(HLabel.MAJOR)
51
52         with mp_hands.Hands(max_num_hands=2, min_detection_confidence=0.5, min_tracking_confidence=0.5) as hands:
53             while GestureController.cap.isOpened() and GestureController.gc_mode:
54                 success, image = GestureController.cap.read()
55
56                 if not success:
57                     print("Ignoring empty camera frame.")
58                     continue
59
60                 image = cv2.cvtColor(cv2.flip(image, flipCode: 1), cv2.COLOR_BGR2RGB)
61                 image.flags.writeable = False
62                 results = hands.process(image)
63
64                 image.flags.writeable = True
65                 image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
66
67                 if results.multi_hand_landmarks:
68                     GestureController.classify_hands(results)
69                     handmajor.update_hand_result(GestureController.hr_major)
70                     handminor.update_hand_result(GestureController.hr_minor)
71
72                     handmajor.set_finger_state()
73                     handminor.set_finger_state()
74                     gest_name = handminor.get_gesture()
75
76                     if gest_name == Gest.PINCH_MINOR:
77                         Controller.handle_controls(gest_name, handminor.hand_result)
78                     else:
79                         gest_name = handmajor.get_gesture()
80                         Controller.handle_controls(gest_name, handmajor.hand_result)
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
509:1 CRLF UTF-8 4 spaces Python 3.10 (pythonProject) ⌂

```



6.2 Output window:

❖ Figure 1. Neutral Gesture



❖ Figure 2. Move Cursor

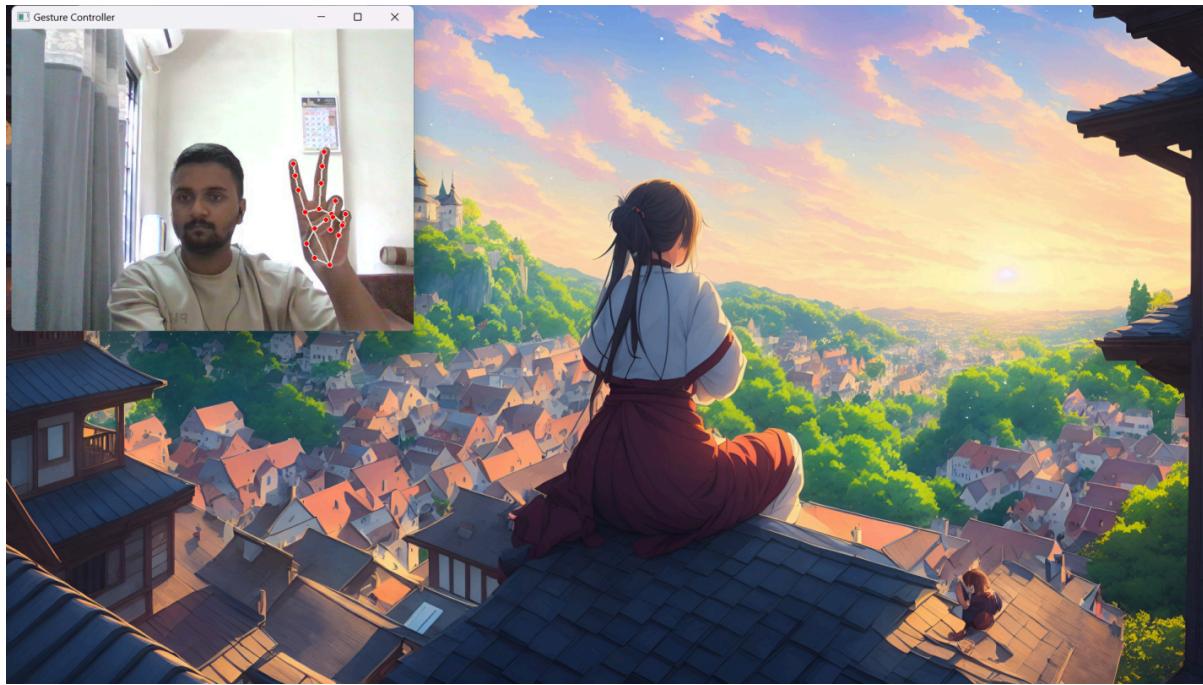


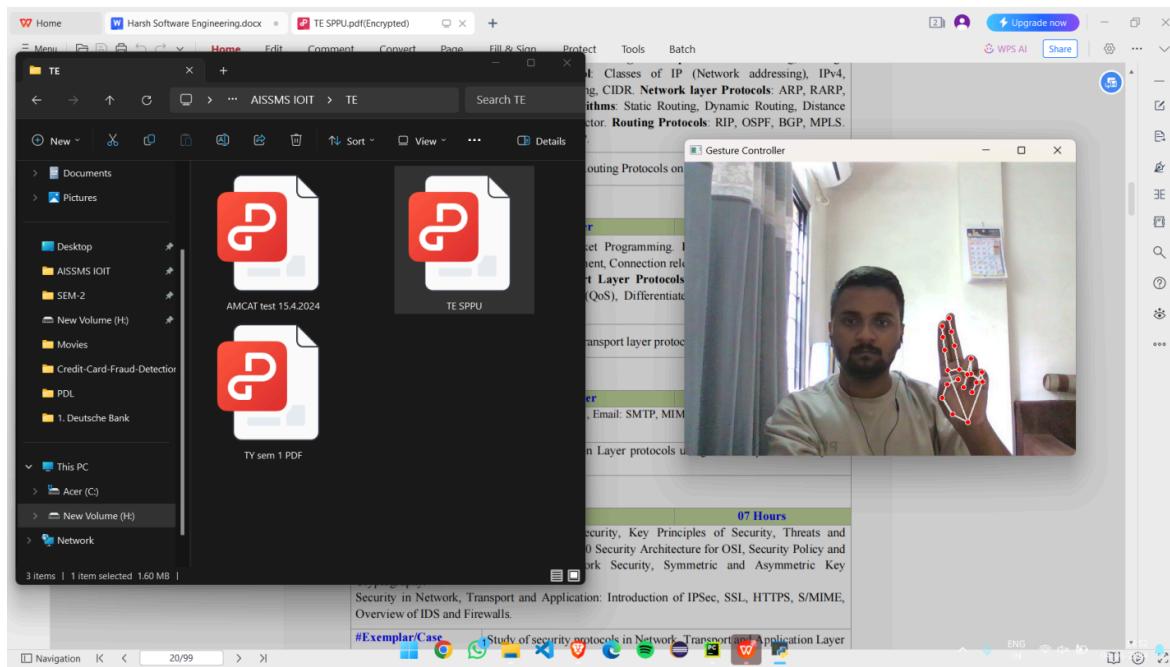
Figure 3. Left Click



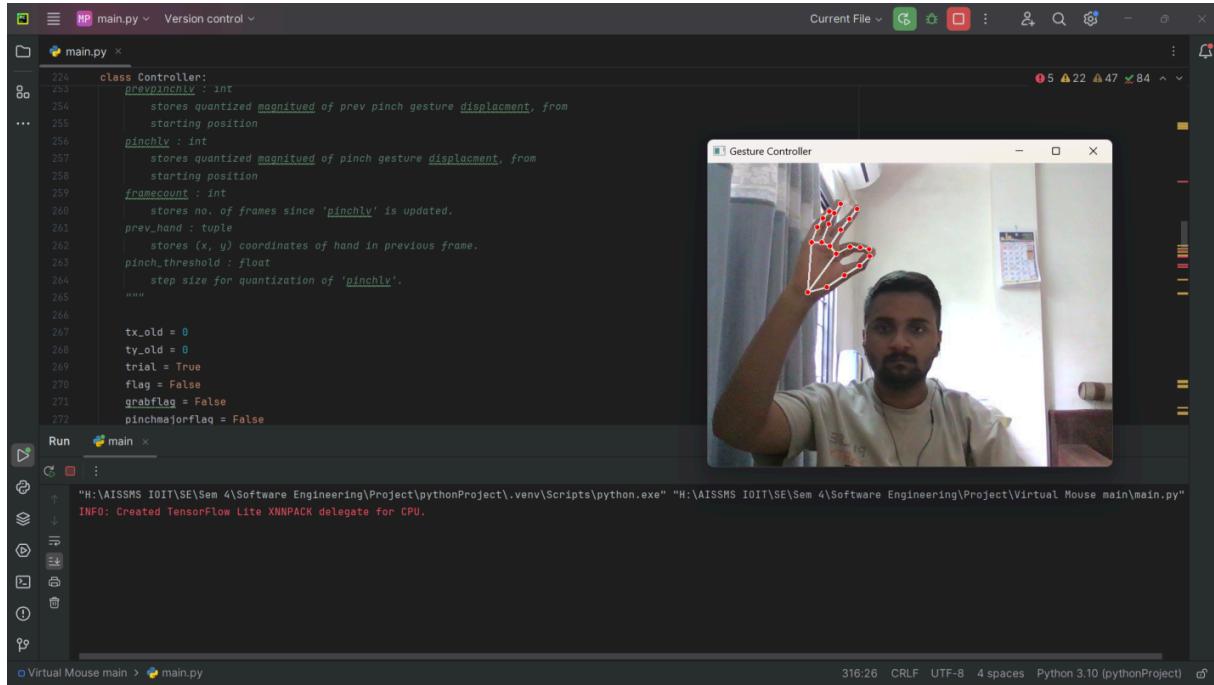
❖ Figure 4. Right Click



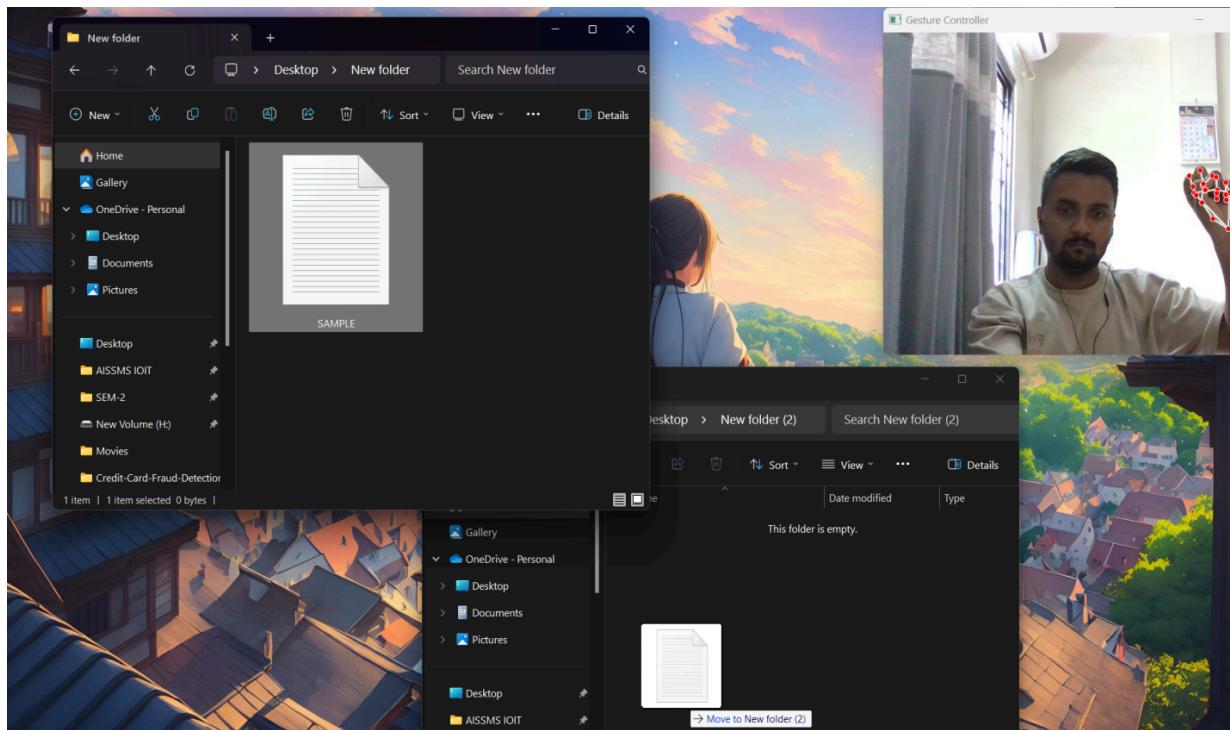
❖ Figure 5. Double Click



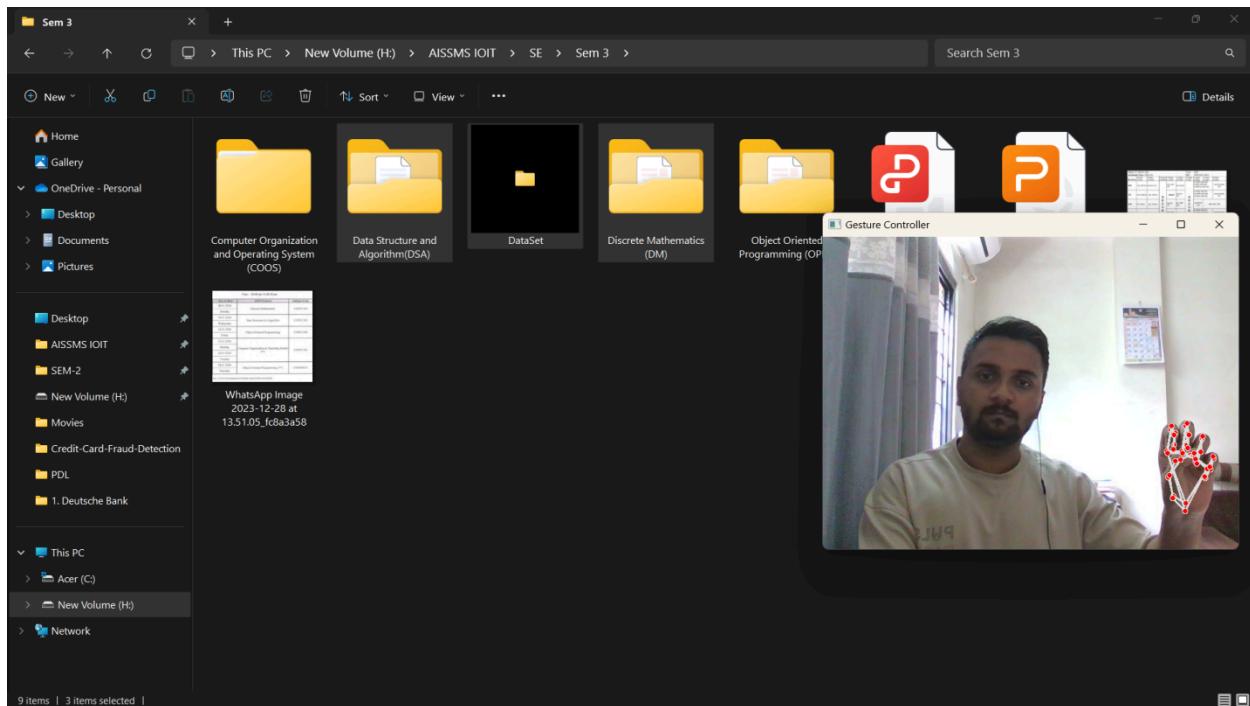
❖ Figure 6. Scrolling



❖ Figure 7. Drag and Drop



❖ **Figure 8. Multiple Item Selection**



CHAPTER 7 : CONCLUSION AND REFERENCES

7.1 Conclusion

This project successfully implements a real-time hand gesture recognition and control system using Python, Mediapipe, and PyAutoGUI. The system leverages Mediapipe's advanced hand-tracking capabilities to detect and interpret a variety of hand gestures, translating them into computer control functions. Key functionalities include cursor movement, clicking actions, adjusting screen brightness, and changing system volume.

The implementation demonstrates the effective use of gesture recognition technology to create an intuitive and natural user interface. By detecting multiple hand gestures and mitigating noise, the system ensures stable and reliable performance. This stability is crucial for practical applications where precise control is necessary.

The project highlights several important aspects:

1. Gesture Detection and Interpretation: The system can recognize a range of gestures, including fist, palm, thumb, and specific finger combinations. These gestures are mapped to different control actions, enhancing the versatility of the interface.
2. Noise Mitigation: By implementing measures to handle fluctuations due to noise, the system maintains accuracy and responsiveness. This is essential for providing a smooth user experience.
3. Multi-Hand Detection: The ability to distinguish between major and minor hands allows for more complex interactions. For instance, different gestures can be assigned to different hands, enabling simultaneous control of multiple functions.
4. Practical Applications: The system's ability to control brightness and volume, navigate through content, and perform mouse actions showcases its potential for various applications. This can significantly enhance accessibility for users with physical disabilities, improve productivity in professional settings, and offer a novel way of interacting with digital devices.



5. Integration of Technologies: The seamless integration of multiple libraries and tools (Mediapipe for hand tracking, PyAutoGUI for control actions, Pycaw for audio control, and Screen Brightness Control for brightness adjustments) demonstrates the power of combining existing technologies to create new solutions.

Overall, this project illustrates the potential of gesture recognition systems to revolutionize human-computer interaction. It paves the way for more natural and intuitive interfaces, making technology more accessible and user-friendly. Future enhancements could include expanding the range of gestures, improving detection accuracy, and exploring additional control applications. This project serves as a strong foundation for further exploration and development in the field of gesture-based control systems.

7.2 References

- OpenCV (cv2): <https://opencv.org/>
- MediaPipe (mp): <https://ai.google.dev/edge/mediapipe/solutions/guide>
- pyautogui: <https://pyautogui.readthedocs.io/>
- Enum: <https://enum-tools.readthedocs.io/en/latest/api/documentation.html>
- ctypes: <https://docs.python.org/3/library/ctypes.html>
- comtypes: <https://pythonhosted.org/comtypes/>
- pycaw: <https://github.com/AndreMiras/pycaw>
- google.protobuf:
<https://developers.google.com/google-ads/api/docs/client-libs/dotnet/working-with-protobuf>