



GESTURE MATE

SRIRAM V.S - 2II4I9I04267

RAGHUL SADAGOBAN S - 2II4I9I04207

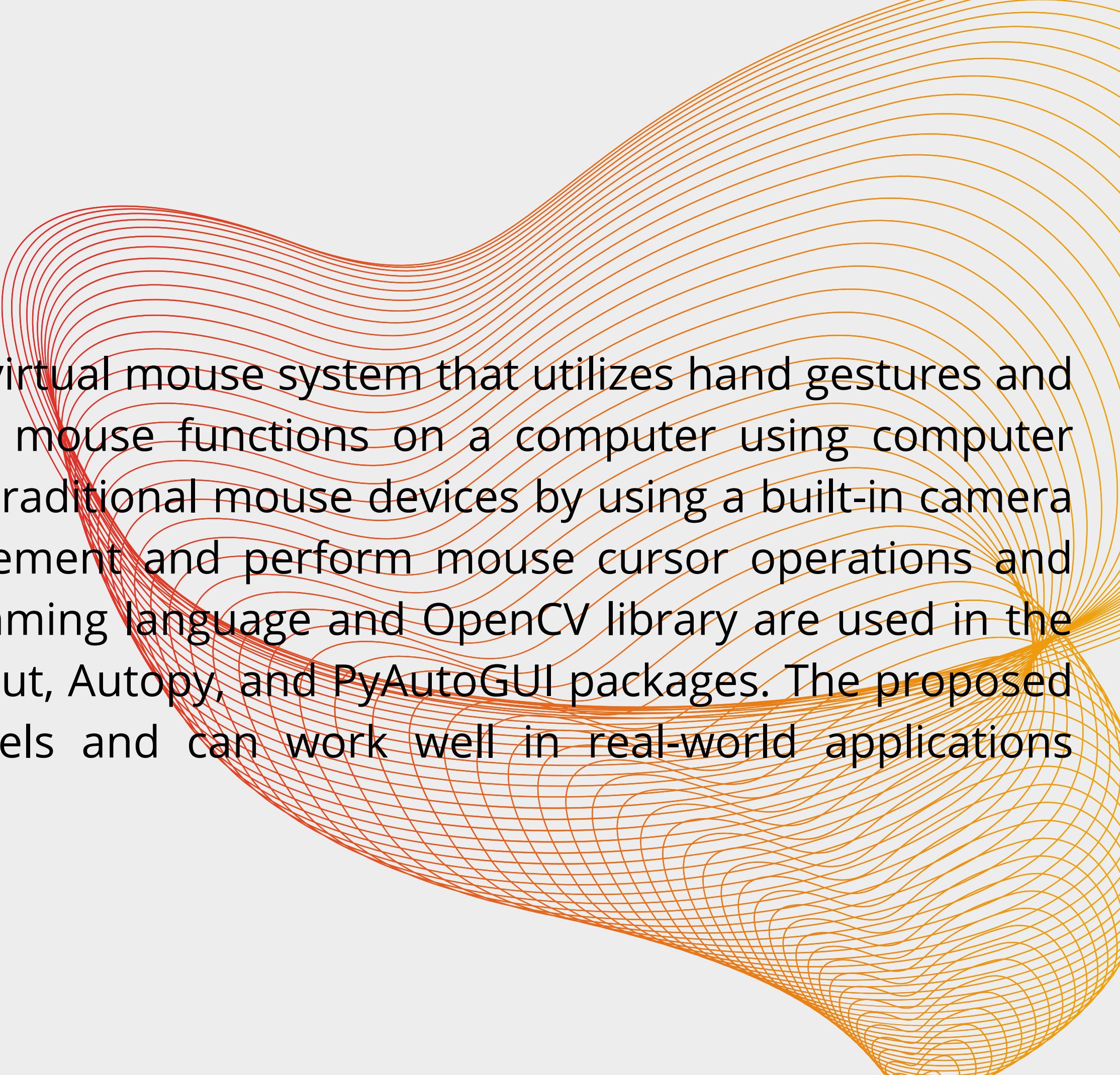
YAZHINIA KUMARAN I - 2II4I9I043I3

Guide Name :- Mrs. A. KANCHANA M.E Assistant Professor



Introduction:

The paper proposes an AI virtual mouse system that utilizes hand gestures and hand tip detection for performing mouse functions on a computer using computer vision. The system aims to replace traditional mouse devices by using a built-in camera or webcam to track fingertip movement and perform mouse cursor operations and scrolling functions. Python programming language and OpenCV library are used in the system, along with MediaPipe, Pyautogui, Autopy, and PyAutoGUI packages. The proposed model achieves high accuracy levels and can work well in real-world applications without the use of a GPU.

An abstract graphic element in the background, featuring a series of thin, orange, wavy lines that curve and flow from the bottom right towards the top left, creating a sense of motion and depth.



Literature Survey:

- This paper presents a study of gesture-based interaction techniques for virtual reality, including virtual mouse control. The authors evaluated different hand gesture recognition techniques and compared their performance in terms of accuracy and usability. They also discuss the design considerations for developing gesture-based interaction systems in virtual reality.
- Title: A Vision-Based Approach for Hand Gesture Recognition in Virtual Mouse Control Author: C. Wang and W. Li Year: 2020 Paper: Journal of Computer Applications, vol. 40, no. 5, pp. 1276-1282.
- This paper presents a vision-based approach for hand gesture recognition in virtual mouse control. The system uses a webcam to capture hand movements and translate them into mouse movements. The authors evaluated the system with a group of users and found that it was accurate and easy to use, demonstrating the potential of this approach for practical applications.



- This paper presents a study of gesture-based interaction techniques for virtual reality, including virtual mouse control. The authors evaluated different hand gesture recognition techniques and compared their performance in terms of accuracy and usability. They also discuss the design considerations for developing gesture-based interaction systems in virtual reality.
- Title: A Vision-Based Approach for Hand Gesture Recognition in Virtual Mouse Control
Author: C. Wang and W. Li Year: 2020 Paper: Journal of Computer Applications, vol. 40, no. 5, pp. 1276-1282.
- This paper presents a vision-based approach for hand gesture recognition in virtual mouse control. The system uses a webcam to capture hand movements and translate them into mouse movements. The authors evaluated the system with a group of users and found that it was accurate and easy to use, demonstrating the potential of this approach for practical applications.



Problem Statement:

1. Traditional methods of controlling slides, such as using a physical mouse or keyboard, can be cumbersome and distracting for the presenter.
2. Gesture control technology can provide a more intuitive and seamless way to navigate through slides during presentations.
3. Current gesture control technology can be expensive and not accessible to everyone.
4. The development of an affordable and user-friendly gesture-controlled virtual mouse for slide presentations would improve the overall user experience.
5. The final product should be easy to set up and use, with clear instructions and minimal technical requirements.



Technology Stack:

HardWare:

1. Webcam
- 2.laptop or Pc

SoftWare:

1. Python
- 2.Anconda
3. VS code
4. Javascript

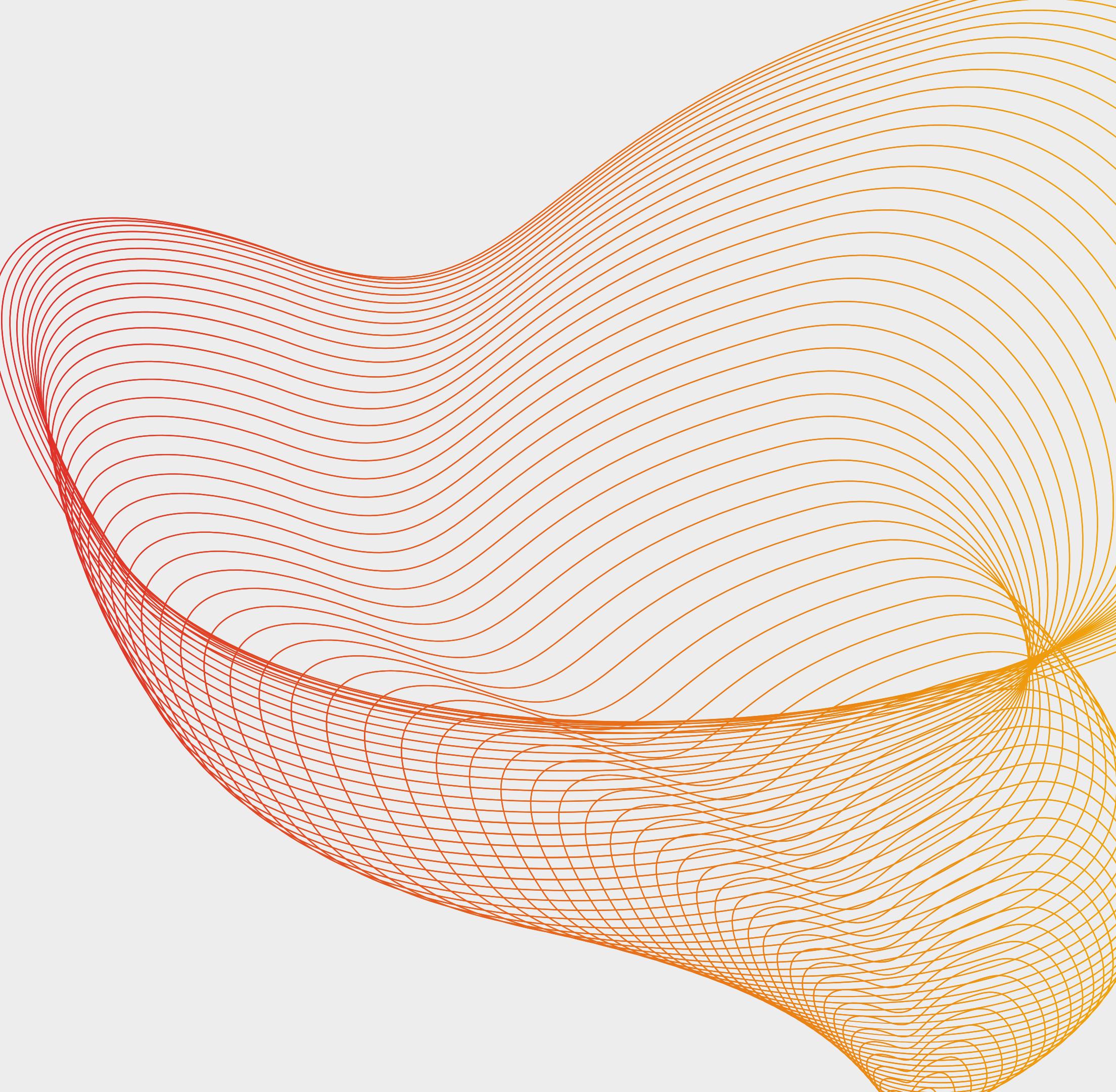
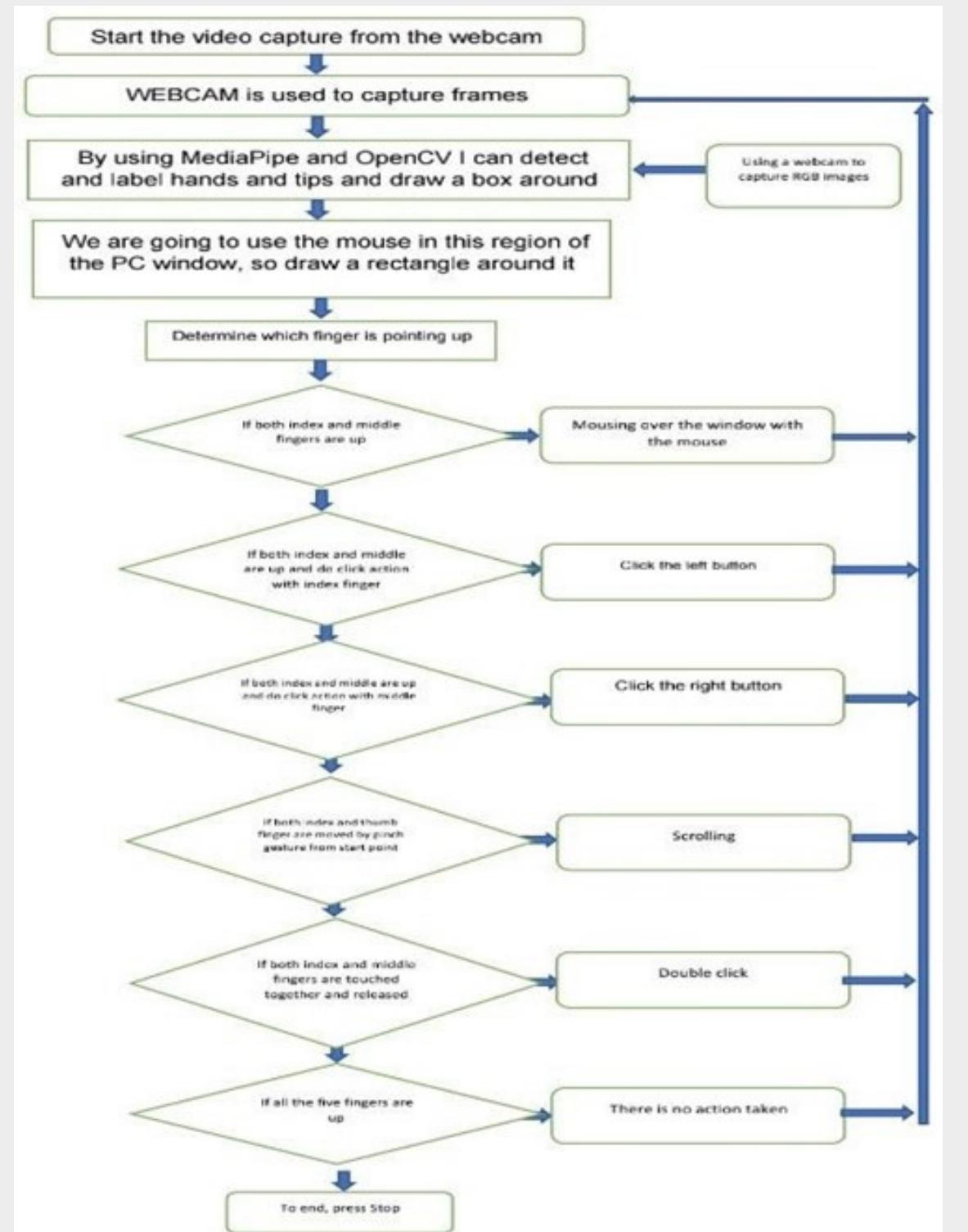


System Architecture:

1. Input Sensors: The system will use sensors, such as a camera or a depth sensor, to capture user hand movements and gestures.
2. Gesture Recognition: The captured hand movements and gestures will be analyzed and recognized by a gesture recognition module. This module will classify the gestures into different categories such as scroll, click, drag, etc.
3. Virtual Mouse Control: Based on the recognized gesture, the system will move the virtual mouse on the screen accordingly. The virtual mouse control module will generate the corresponding mouse events such as left-click, right-click, scroll, drag, etc.
4. Feedback Mechanism: The system will provide feedback to the user, indicating the success or failure of the recognition of the gesture. The feedback can be in the form of a sound, vibration, or a visual indicator.



System Design:





Module Description:

1. Hand Detection:

- Encouraging the discovery and following of hands and separating them from the environment is important.
- Managing fabric objects can interfere with the ability to see and distinguish hands.
- Differences in appearance and non-biological factors can cause issues.

2. Hand Tracking:

- Tracking the hands is reliable at close range, but visibility may become difficult in virtual reality.
- As virtual reality expands, hand tracking will become more reliable.



3. Involvement and Inclusivity:

- Hand tracking and demonstrations pose challenges related to irregular embodiment and inclusion.
- Separating the skin from the surrounding area is crucial for hand tracking and visualization.
- Skin color is a potential issue that is not explicitly addressed in hand tracking technology.

4. Gesture Recognition Based on Computer Vision:

- Touch Image Collection: The touch-based touch perception method involves collecting touch image information from one or more cameras.
- Pre-processing: The collected data is pre-processed, including sound removal and information enhancement.



- Separation Algorithm: A separation algorithm is used to detect the target touch in the image.
- Touch Recognition: Touch-based touch recognition consists of three main parts: touch recognition, touch analysis, and touch detection.
- Input Image Segmentation: The first step in touch recognition is to tap the input image to create a segment.
- Touch Separation: The touch separation process consists of two main parts: touch area processing and touch separation.
- Touch Area Processing: The touch area process removes the touch circuit from the background complex with self-tracking of the image containing the touch.
- Touch Separation: The touch separation phase segment uses a post-touch algorithm to separate the current touch from the background.
- Touch Modelling: Touch modelling technologies mainly include visual-based touch modelling and 3D model-based touch models.



Performance Evaluation:

An AI system utilizing computer vision to improve human-computer interaction is proposed in the proposed visual mouse AI system. Because of the limited number of data sets available, it is difficult to compare contradictory outcomes of testing of visual AI mouse systems. A total of 600 hand-labeled touches were obtained through this test, which was performed 25 times by four people in different lighting conditions and at different distances from the screen, and each person tested the visual mouse system individually. In Table 1, the results of 10 AI tests performed in normal light conditions, 5 AI tests performed in dim light, 5 AI tests performed near the webcam, and 5 AI tests performed at a distance from the webcam are tabulated.



Table:

Hand tip gesture * performed		Success	Failure	
	(%)			
Tip ID 1 or both tip IDs 1 and 2 are up		Mouse movement	100	0
Tip IDs 0 and 1 are up and the distance between the fingers is <30		Left button click	99	1
Tip IDs 1 and 2 are up and the distance between the fingers is <40		Right button click	95	5
Tip IDs 1 and 2 are up and the distance between the fingers is >40 and both fingers are moved up the page		Scroll up function	100	0
Tip IDs 1 and 2 are up and the distance between the fingers is >40 and both fingers are moved down the page		Scroll down function	100	0
All five tip IDs 0, 1, 2, 3, and 4 are up		No action performed	100	0
Result			594	6

* Finger tip ID for respective fingers: tip Id 0: thumb finger; tip Id 1: index finger; tip Id 2: middle finger; tip Id 3: ring finger; tip Id 4: little finger.

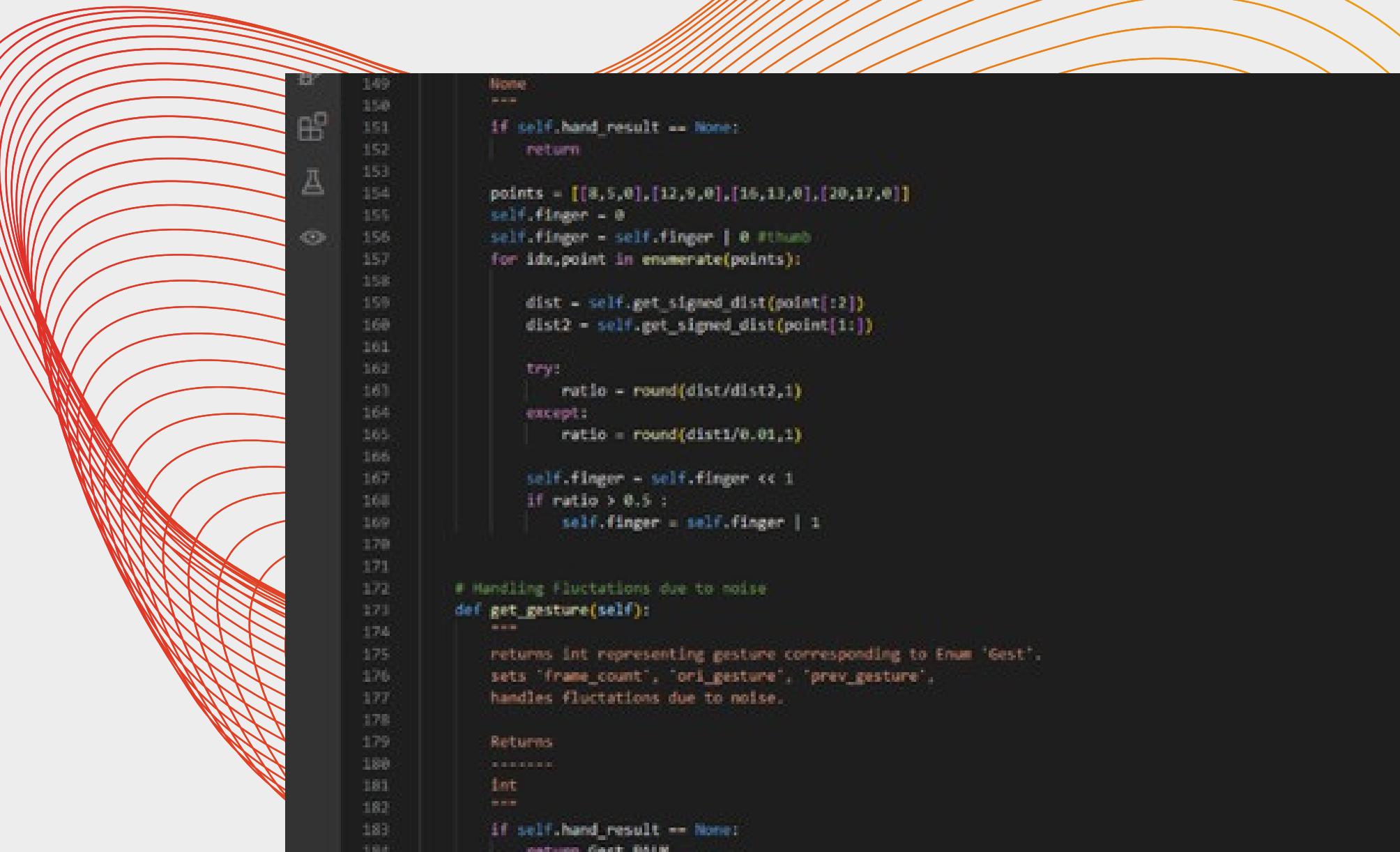


Coding Screen Shots:

File Edit Selection View Go Run Terminal Help Gesture_Controller.py - Visual Studio Code

```
Gesture_Controller.py M X
G > Users > silas > Gesture-Controlled-Virtual-Mouse > src > Gesture_Controller.py

212     self.frame_count = 0
213
214     self.prev_gesture = current_gesture
215
216     if self.frame_count > 4 :
217         self.ori_gesture = current_gesture
218     return self.ori_gesture
219
220 # Executes commands according to detected gestures
221 class Controller:
222     """
223     Executes commands according to detected gestures.
224
225     Attributes
226     -----
227     tx_old : int
228         previous mouse location x coordinate
229     ty_old : int
230         previous mouse location y coordinate
231     flag : bool
232         true if V gesture is detected
233     grabFlag : bool
234         true if FIST gesture is detected
235     pinchmajorflag : bool
236         true if PINCH gesture is detected through MAJOR hand,
237         on x-axis 'Controller.changesystembrightness',
238         on y-axis 'Controller.changesystemvolume'.
239     pinchminorflag : bool
240         true if PINCH gesture is detected through MINOR hand,
241         on x-axis 'Controller.scrollHorizontal',
242         on y-axis 'Controller.scrollVertical'.
243     pinchstartxcoord : int
244         x coordinate of hand landmark when pinch gesture is started.
245     pinchstartycoord : int
246         y coordinate of hand landmark when pinch gesture is started.
247     pinchdirectionflag : bool
248         true if pinch gesture movement is along x-axis,
249         otherwise false
250     prevpinchlv : int
251         stores quantized magnitude of prev pinch gesture displacement, from
252         starting position
253     pinchlv : int
254         stores quantized magnitude of pinch gesture displacement, from
255         starting position
256     framecount : int
257         stores no. of frames since 'pinchlv' is updated.
258     prev_hand : tuple
259         stores (x, y) coordinates of hand in previous frame.
```



```
147     None
148
149     if self.hand_result == None:
150         return
151
152     points = [(8,5,0),(12,9,0),(16,13,0),(20,17,0)]
153     self.finger = 0
154     self.finger = self.finger | 0 #init
155     for idx,point in enumerate(points):
156
157         dist = self.get_signed_dist(point[:2])
158         dist2 = self.get_signed_dist(point[1:])
159
160         try:
161             ratio = round(dist/dist2,1)
162         except:
163             ratio = round(dist/0.01,1)
164
165         self.finger = self.finger << 1
166         if ratio > 0.5 :
167             self.finger = self.finger | 1
168
169
170     # Handling Fluctuations due to noise
171     def get_gesture(self):
172         """
173
174             returns int representing gesture corresponding to Enum 'Gest'.
175             sets 'frame_count', 'ori_gesture', 'prev_gesture',
176             handles fluctuations due to noise.
177
178
179         Returns
180         -----
181         int
182         ---
183
184         if self.hand_result == None:
185             return Gest.noise
```



Coding Screen Shots:

File Edit Selection View Go Run Terminal Help

Gesture_Controller.py - Visual Studio Code

```
C:\Users\slas> Gesture-Controlled-Virtual-Mouse > src > Gesture_Controller.py
```

```
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     return (x,y)
362
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.pinchlvl = 0
368     Controller.prevpinchlvl = 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 'pinchlvl'.
376
377             Parameters
378             -----
379             hand_result : Object
380                 Landmarks obtained from mediapipe.
381             controlHorizontal : callback function assosiated with horizontal
382                 pinch gesture.
383             controlVertical : callback function assosiated with vertical
384                 pinch gesture.
385
386             Returns
387             -----
388             None
389             """
390             if Controller.framecount == 5:
391                 Controller.framecount = 0
392                 Controller.pinchlvl = Controller.prevpinchlvl
```

File Edit Selection View Go Run Terminal Help

Gesture_Controller.py - Visual Studio Code

```
C:\Users\slas> Gesture-Controlled-Virtual-Mouse > src > Gesture_Controller.py
```

```
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     return (x,y)
362
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.pinchlvl = 0
368     Controller.prevpinchlvl = 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 'pinchlvl'.
376
377             Parameters
378             -----
379             hand_result : Object
380                 Landmarks obtained from mediapipe.
381             controlHorizontal : callback function assosiated with horizontal
382                 pinch gesture.
383             controlVertical : callback function assosiated with vertical
384                 pinch gesture.
385
386             Returns
387             -----
388             None
389             """
390             if Controller.framecount == 5:
391                 Controller.framecount = 0
392                 Controller.pinchlvl = Controller.prevpinchlvl
```



Coding Screen Shots:

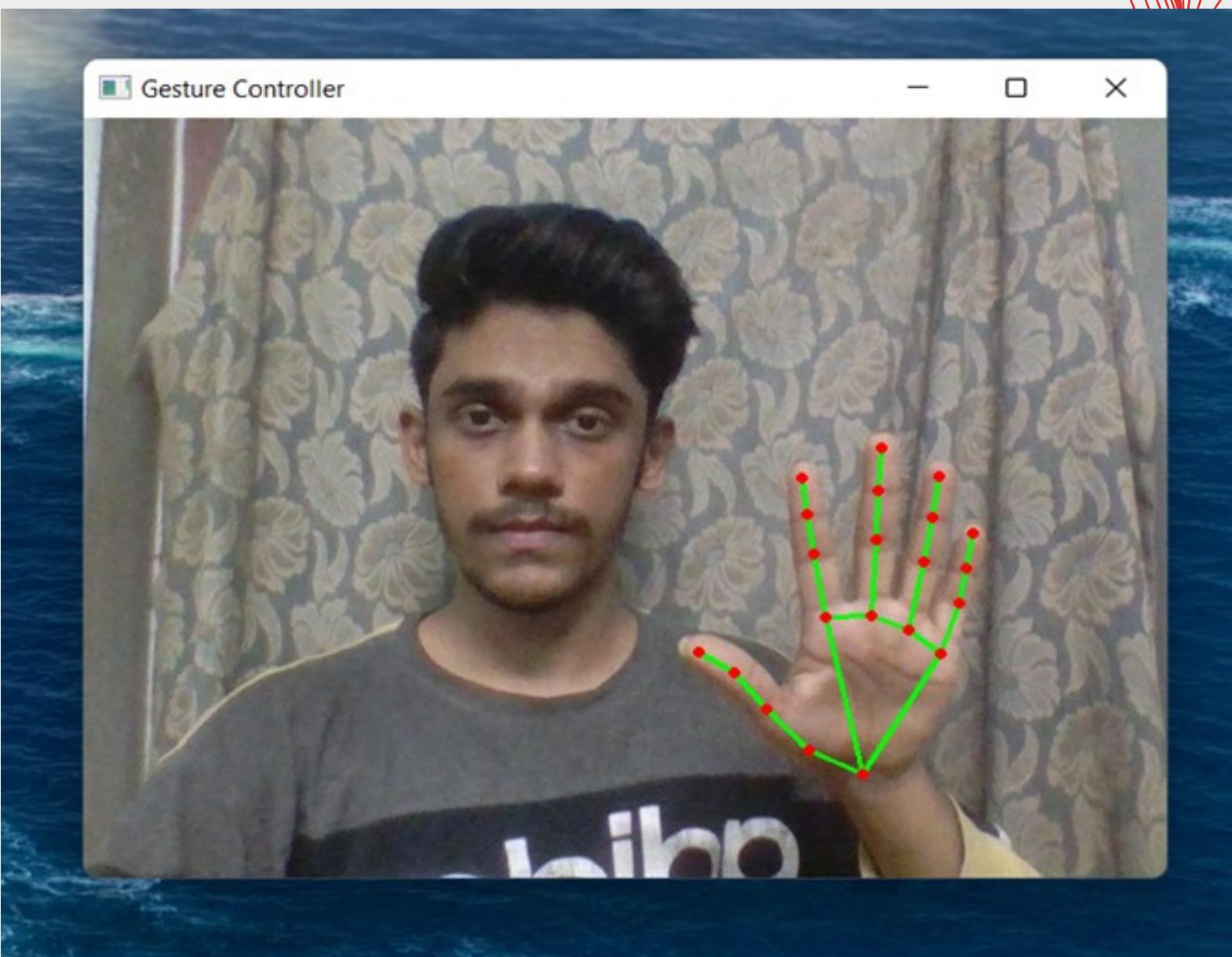
```
File Edit Selection View Go Run Terminal Help
Gesture_Controller_Gloved.py M Gesture_Controller_Gloved.py X
C:\Users\slas>Gesture-Controlled-Virtual-Mouse>src> Gesture_Controller_Gloved.py
133     except:
134         slope_12 = (c1[1]-c2[1])*999.0 + 0.1
135
136     try:
137         slope_14 = -1 / slope_12
138     except:
139         slope_14 = -999.0
140
141     if slope_14 < 0:
142         sign = 1
143     else:
144         sign = -1
145
146     bot_rx = int(cx + self.roi_alpha2 * 1 * np.sqrt(1/(1+slope_12**2)))
147     bot_ry = int(cy + self.roi_alpha2 * slope_12 * 1 * np.sqrt(1/(1+slope_12**2)))
148
149     bot_lx = int(cx - self.roi_alpha1 * 1 * np.sqrt(1/(1+slope_12**2)))
150     bot_ly = int(cy - self.roi_alpha1 * slope_12 * 1 * np.sqrt(1/(1+slope_12**2)))
151
152     top_lx = int(bot_lx + sign * self.roi_beta * 1 * np.sqrt(1/(1+slope_14**2)))
153     top_ly = int(bot_ly + sign * self.roi_beta * slope_14 * 1 * np.sqrt(1/(1+slope_14**2)))
154
155     top_rx = int(bot_rx + sign * self.roi_beta * 1 * np.sqrt(1/(1+slope_14**2)))
156     top_ry = int(bot_ry + sign * self.roi_beta * slope_14 * 1 * np.sqrt(1/(1+slope_14**2)))
157
158     bot_lx = in_cam(bot_lx, 'x')
159     bot_ly = in_cam(bot_ly, 'y')
160
161     bot_rx = in_cam(bot_rx, 'x')
162     bot_ry = in_cam(bot_ry, 'y')
163
164     top_lx = in_cam(top_lx, 'x')
165     top_ly = in_cam(top_ly, 'y')
166
167     top_rx = in_cam(top_rx, 'x')
168     top_ry = in_cam(top_ry, 'y')
169
170     self.roi_corners = [(bot_lx,bot_ly), (bot_rx,bot_ry), (top_rx,top_ry), (top_lx,top_ly)]
171
172
173 def find_glove_hsv(self, frame, marker):
174     rec_coor = marker.corners[0][0]
175     c1 = (int(rec_coor[0][0]),int(rec_coor[0][1]))
176     c2 = (int(rec_coor[1][0]),int(rec_coor[1][1]))
177     c3 = (int(rec_coor[2][0]),int(rec_coor[2][1]))
178     c4 = (int(rec_coor[3][0]),int(rec_coor[3][1]))
179
180     l = np.absolute(ecu_dis(c1,c4))
181
182     if l > 0:
183         slope_12 = (c1[1]-c2[1])/(c1[0]-c2[0])
184         slope_14 = (c1[1]-c3[1])/(c1[0]-c3[0])
185
186         if slope_12 < 0:
187             sign = 1
188         else:
189             sign = -1
190
191         bot_rx = int(cx + self.roi_alpha2 * 1 * np.sqrt(1/(1+slope_12**2)))
192         bot_ry = int(cy + self.roi_alpha2 * slope_12 * 1 * np.sqrt(1/(1+slope_12**2)))
193
194         bot_lx = int(cx - self.roi_alpha1 * 1 * np.sqrt(1/(1+slope_12**2)))
195         bot_ly = int(cy - self.roi_alpha1 * slope_12 * 1 * np.sqrt(1/(1+slope_12**2)))
196
197         top_lx = int(bot_lx + sign * self.roi_beta * 1 * np.sqrt(1/(1+slope_14**2)))
198         top_ly = int(bot_ly + sign * self.roi_beta * slope_14 * 1 * np.sqrt(1/(1+slope_14**2)))
199
200         top_rx = int(bot_rx + sign * self.roi_beta * 1 * np.sqrt(1/(1+slope_14**2)))
201         top_ry = int(bot_ry + sign * self.roi_beta * slope_14 * 1 * np.sqrt(1/(1+slope_14**2)))
202
203         bot_lx = in_cam(bot_lx, 'x')
204         bot_ly = in_cam(bot_ly, 'y')
205
206         bot_rx = in_cam(bot_rx, 'x')
207         bot_ry = in_cam(bot_ry, 'y')
208
209         top_lx = in_cam(top_lx, 'x')
210         top_ly = in_cam(top_ly, 'y')
211
212         top_rx = in_cam(top_rx, 'x')
213         top_ry = in_cam(top_ry, 'y')
214
215         self.roi_corners = [(bot_lx,bot_ly), (bot_rx,bot_ry), (top_rx,top_ry), (top_lx,top_ly)]
```

```
File Edit Selection View Go Run Terminal Help
Gesture_Controller_Gloved.py M Gesture_Controller_Gloved.py X
C:\Users\slas>Gesture-Controlled-Virtual-Mouse>src> Gesture_Controller_Gloved.py
88     elif type_ == 'y':
89         if val<0:
90             return 0
91         if val>GestureController.cam_height:
92             return GestureController.cam_height
93         return val
94
95
96 class ROI:
97     def __init__(self, roi_alpha1=1.5, roi_alpha2=1.5, roi_beta=2.5, hsv_alpha = 0.3, hsv_beta = 0.5, hsv_lift_up = 0.3):
98         self.roi_alpha1 = roi_alpha1
99         self.roi_alpha2 = roi_alpha2
100        self.roi_beta = roi_beta
101        self.roi_corners = None
102
103        self.hsv_alpha = hsv_alpha
104        self.hsv_beta = hsv_beta
105        self.hsv_lift_up = hsv_lift_up
106        self.hsv_corners = None
107
108        self.marker_top = None
109        self.glove_hsv = None
110
111    def findROI(self, frame, marker):
112        rec_coor = marker.corners[0][0]
113        c1 = (int(rec_coor[0][0]),int(rec_coor[0][1]))
114        c2 = (int(rec_coor[1][0]),int(rec_coor[1][1]))
115        c3 = (int(rec_coor[2][0]),int(rec_coor[2][1]))
116        c4 = (int(rec_coor[3][0]),int(rec_coor[3][1]))
117
118        try:
119            marker.marker_x2y = np.sqrt((c1[0]-c2[0])**2 + (c1[1]-c2[1])**2) / np.sqrt((c3[0]-c2[0])**2 + (c3[1]-c2[1])**2)
120        except:
121            marker.marker_x2y = 999.0
122
123        #mid-point of top line of Marker
124        cx = (c1[0] + c2[0])/2
125        cy = (c1[1] + c2[1])/2
126
127        self.marker_top = [cx, cy]
128
129        l = np.absolute(ecu_dis(c1,c4))
130
131        try:
132            slope_12 = (c1[1]-c2[1])/(c1[0]-c2[0])
133        except:
134            slope_12 = (c1[1]-c2[1])*999.0 + 0.1
135
```

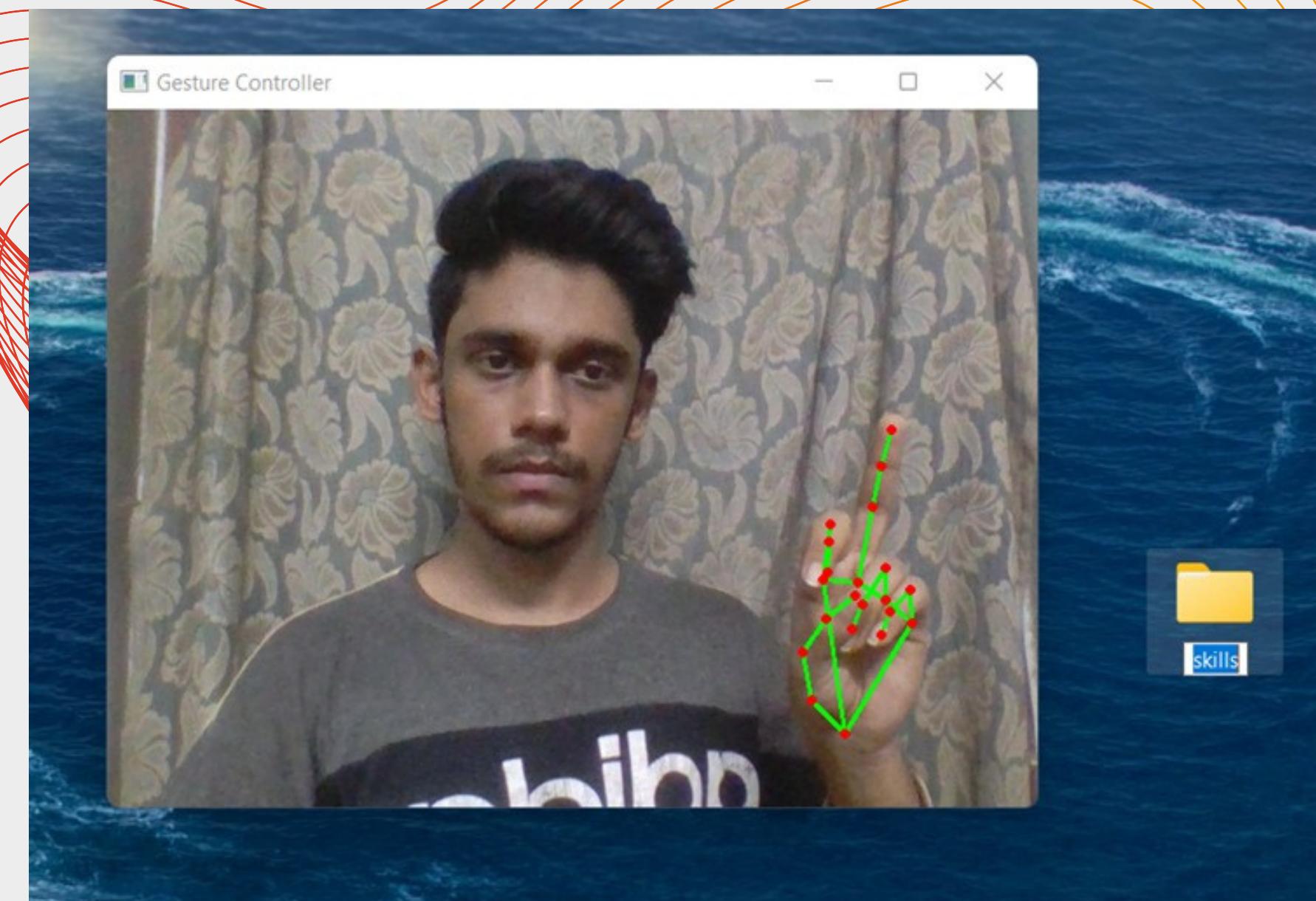


Output Screen Shots:

Neutral Gesture



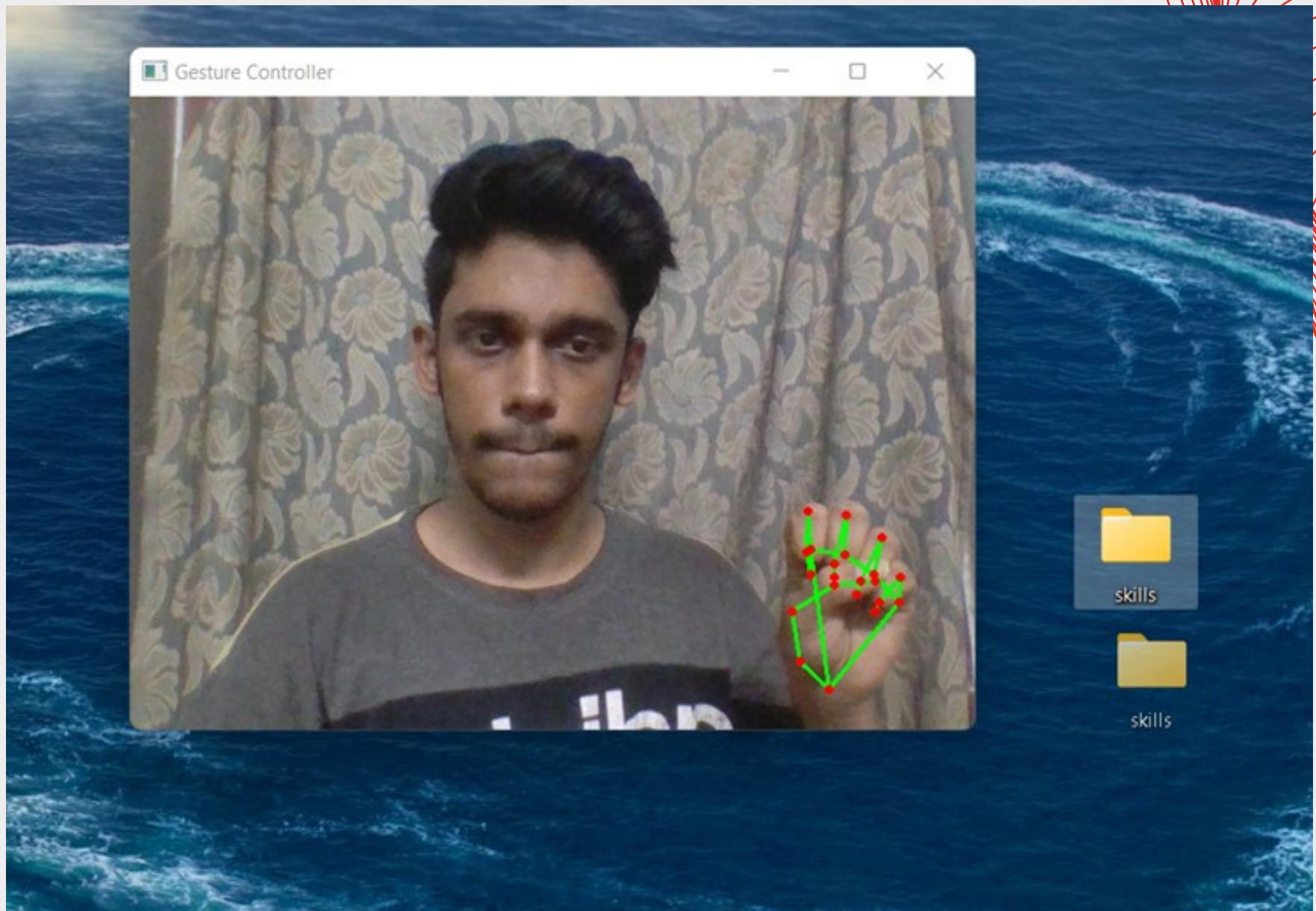
Left Click:



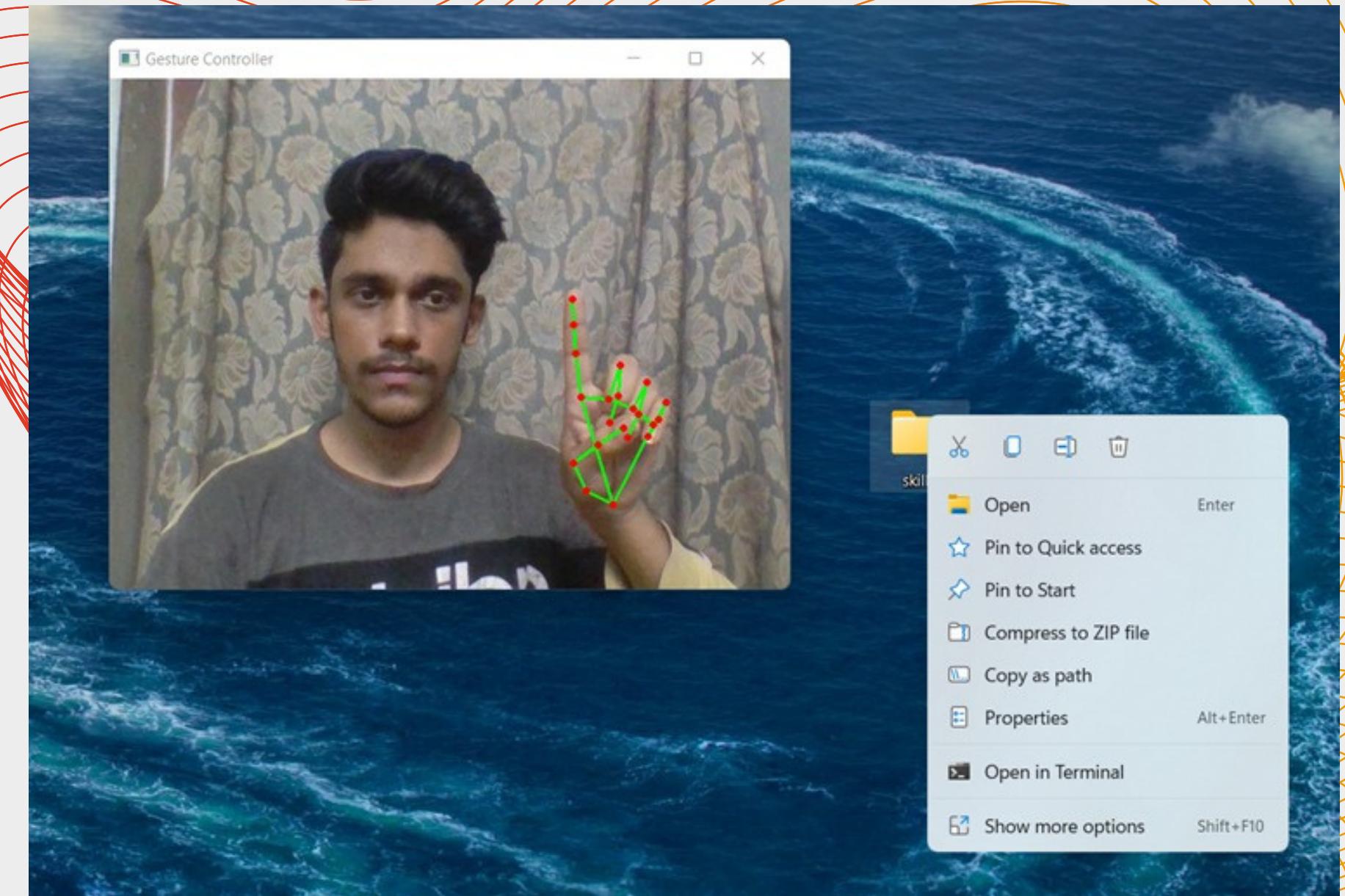


Screen Shots:

Drag and drop:



Right Click:





Conclusion :

- AI virtual mouse system allows controlling mouse cursor functions using hand gestures
- Uses webcam or built-in camera to detect hand gestures and process frames
- Proposed model has higher accuracy compared to existing models
- Can be used for real-world applications and to reduce the spread of COVID-19
- Limitations include decreased accuracy in right-click function and difficulty in clicking and dragging to select text
- Next steps include improving finger tip detection algorithm to address limitations.



References :

- [1] Abhik Banerjee, Abhirup Ghosh, Koustuvmoni Bharadwaj," Mouse Control using a Web Camera based on Color Detection",IJCTT,vol.9, Mar 2014.
- [2] Angel, Neethu.P.S,"Real Time Static & Dynamic Hand Gesture Recognition", International Journal of Scientific & Engineering Research Volume 4, Issue3, March-2013.
- [3] Q. Y. Zhang, F. Chen and X. W. Liu, "Hand Gesture Detection and Segmentation Based on Difference Background Image with Complex Background," Proceedings of the 2008 International Conference on Embedded Software and Systems, Sichuan, 29-31 July 2008, pp. 338-343.



- [4] Inside Facebook Reality Labs: Wrist-based interaction for the next computing platform [WWW Document], 2021 Facebook Technol. URL <https://tech.fb.com/inside-facebook-realitylabs-wrist-based-interaction-for-the-next-computing-platform/> (accessed 3.18.21).
- [5] Danckert, J., Goodale, M.A., 2001. Superior performance for visually guided pointing in the lower visual field. *Exp. Brain Res.* 137, 303–308. <https://doi.org/10.1007/s002210000653>.
- [6] Carlton, B., 2021. Hapt X Launches True-Contact Haptic Gloves For VR And Robotics. VR Scout. URL <https://vrscout.com/news/haptx-truecontact-haptic-gloves-vr/> (accessed 3.10.21).
- [7] Brenton, H., Gillies, M., Ballin, D., Chatting, D., 2005. D.: The uncanny valley: does it exist, in: In: 19th British HCI Group Annual Conference: Workshop on Human-Animated Character Interaction.



Thank You

