

A

Synopsis/Project Report

On

## **Hill Climbing Game With Hand Gestures**

Submitted in partial fulfillment of the requirement for the V semester

**Bachelor of Computer Science and Engineering**

By

**Rohit Chand**

Under the Guidance of

**Dr. Vikrant Sharma**

**Assistant Professor**

**Department of CSE**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**GRAPHIC ERA HILL UNIVERSITY, DEHRADUN CAMPUS**

**2022- 2023**

## **STUDENT'S DECLARATION**

I , **Mr Rohit Chand** here by declare the work,

which is being presented in the project, entitled “**Hill Climbing Game With Hand Gestures**”

in partial fulfillment of the requirement for the award of the degree **B. Tech** in the session **2023-**

**2024**, is an authentic record of my own work carried out under the supervision of “**Dr. Vikrant**

**Sharma**”, Assistant Professor, Department of CSE, Graphic Era Hill University, Dehradun.

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Rohit Chand



## **CERTIFICATE**

The project report entitled “Hill Climbing Game With Hand Gestures” being submitted by  
Rohit Chand to Graphic Era Hill University

Dehradun Campus for the award of Bonafede work carried out by him. He have worked  
under my guidance and supervision and fulfilled the requirement for the submission of  
report.

**Vikrant Sharma**

**Project Guide**

**(HOD, CSE Dept.)**



## **ACKNOWLEDGEMENT**

I take immense pleasure in thanking Honorable **“Dr. Vikrant Sharma”**(Assistant Professor, **GEHU Dehradun Campus**) for permitting me and carrying out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me everything that I need.

I again want to extend thanks to our President **“Prof. (Dr.) Kamal Ghanshala”** for providing us all infrastructure and facilities to work in need without which this work could not be possible.

r Many thanks to Professors and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this thesis.

Finally, yet importantly, I would like to express my heartiest thanks to my beloved parents, for their moral support, affection, and blessings. I would also like to pay our sincere thanks to all our friends and well-wishers for their help and wishes for the successful completion of this research.

**Rohit Chand**

## **TABLE OF CONTENTS**

<b>Declaration.....</b>	
<b>Certificate.....</b>	
<b>Acknowledgement.....</b>	
<b>Abstract.....</b>	
<b>CHAPTER 1:</b>	<b>INTRODUCTION.....</b>
1.1	Background and Motivation.....
1.2	Problem Statement.....
1.3	Objectives .....
1.4	Scope of the Research.....
<b>CHAPTER 2:</b>	<b>LITERATURE REVIEW.....</b>
2.1	Gesture Recognition Techniques.....
2.2	AI-based Human-Computer Interaction.....
2.3	Hand Tracking and Detection.....
2.4	Hand Gesture Recognition Systems.....
2.5	Summary.....
<b>CHAPTER 3:</b>	<b>SYSTEM ARCHITECTURE AND METHODOLOGY</b>
3.1	System Overview.....
3.2	Block Diagram of system.....
3.3	Flow chart of Modules of System.....
<b>CHAPTER 4:</b>	<b>SYSTEM DEVELOPMENT &amp; IMPLEMENTATION.....</b>

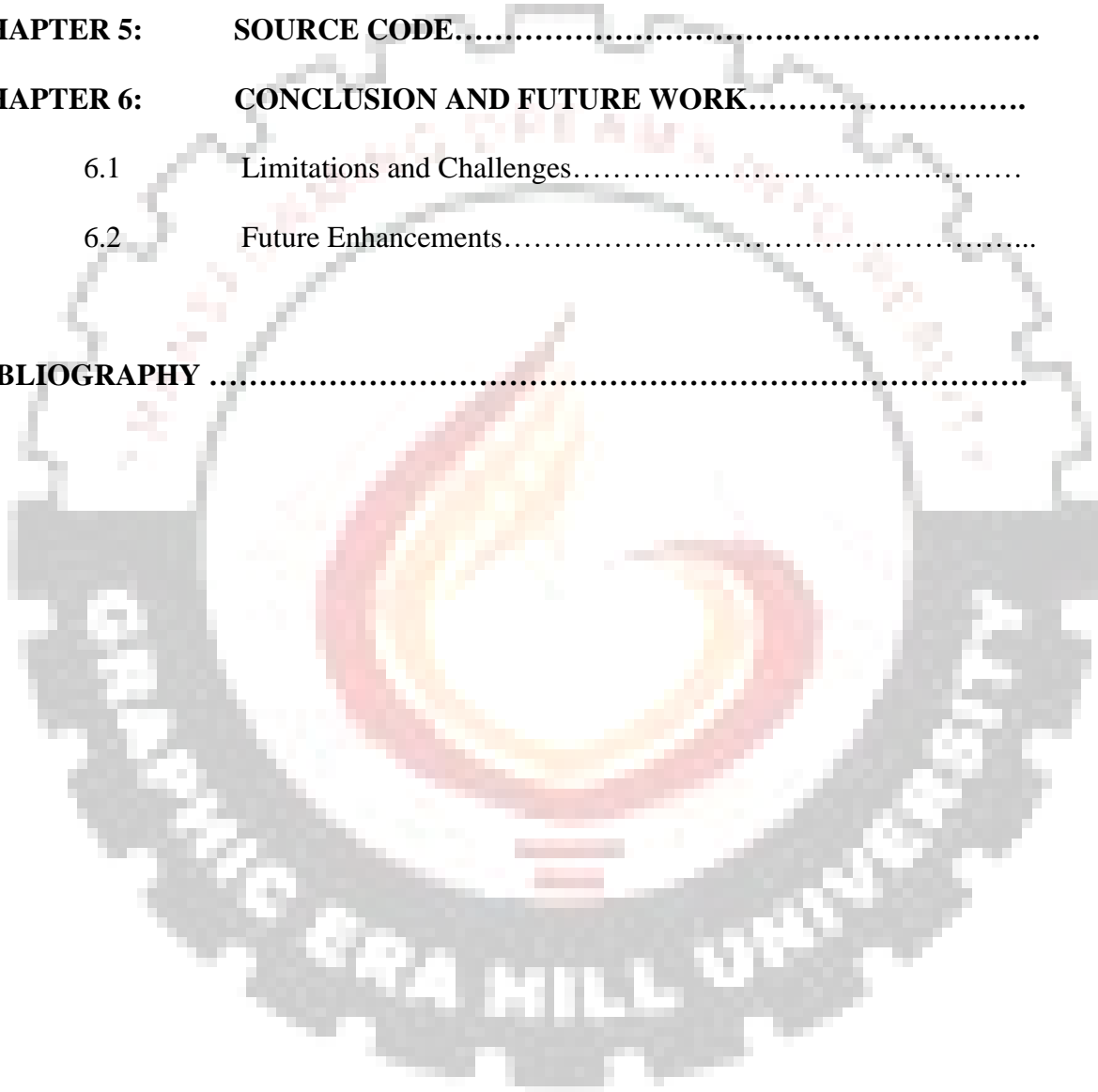
4.1	Tools and Language .....
4.2	Hand Tracking Module.....
4.3	Ai Virtual Mouse Coding.....

**CHAPTER 5: SOURCE CODE.....**

**CHAPTER 6: CONCLUSION AND FUTURE WORK.....**

6.1	Limitations and Challenges.....
6.2	Future Enhancements.....

**BIBLIOGRAPHY .....**



# INTRODUCTION

## 1.1 Background and Motivation

In recent years, AI-based technologies have gained significant popularity and have revolutionized various industries and fields. One area that has seen remarkable advancements is human-computer interaction. Traditional input devices, such as keyboards and mice, have limitations in terms of natural and intuitive interaction. Users often require additional training to master complex interfaces, leading to reduced productivity and user experience.

The motivation behind this research is to develop a more efficient and user-friendly screen controller using AI and hand gestures. By leveraging computer vision and machine learning techniques, the aim is to create a system that allows users to interact with their screens effortlessly, mimicking natural hand movements. This would not only enhance user experience but also open up new possibilities for applications in areas such as gaming, design, and virtual reality.

## 1.2 Problem Statement

The traditional input devices used for screen control have several limitations. Keyboards and mouse, although widely used, may not provide the most intuitive and efficient interaction experience. Users often need to memorize complex shortcuts and gestures, which can be time consuming and prone to errors. There is a need for a more natural and user-friendly interface that simplifies screen control and improves overall productivity.

## 1.3 Objectives

The main objective of this research is to develop an AI-based screen controller using hand gestures.

The specific objectives include

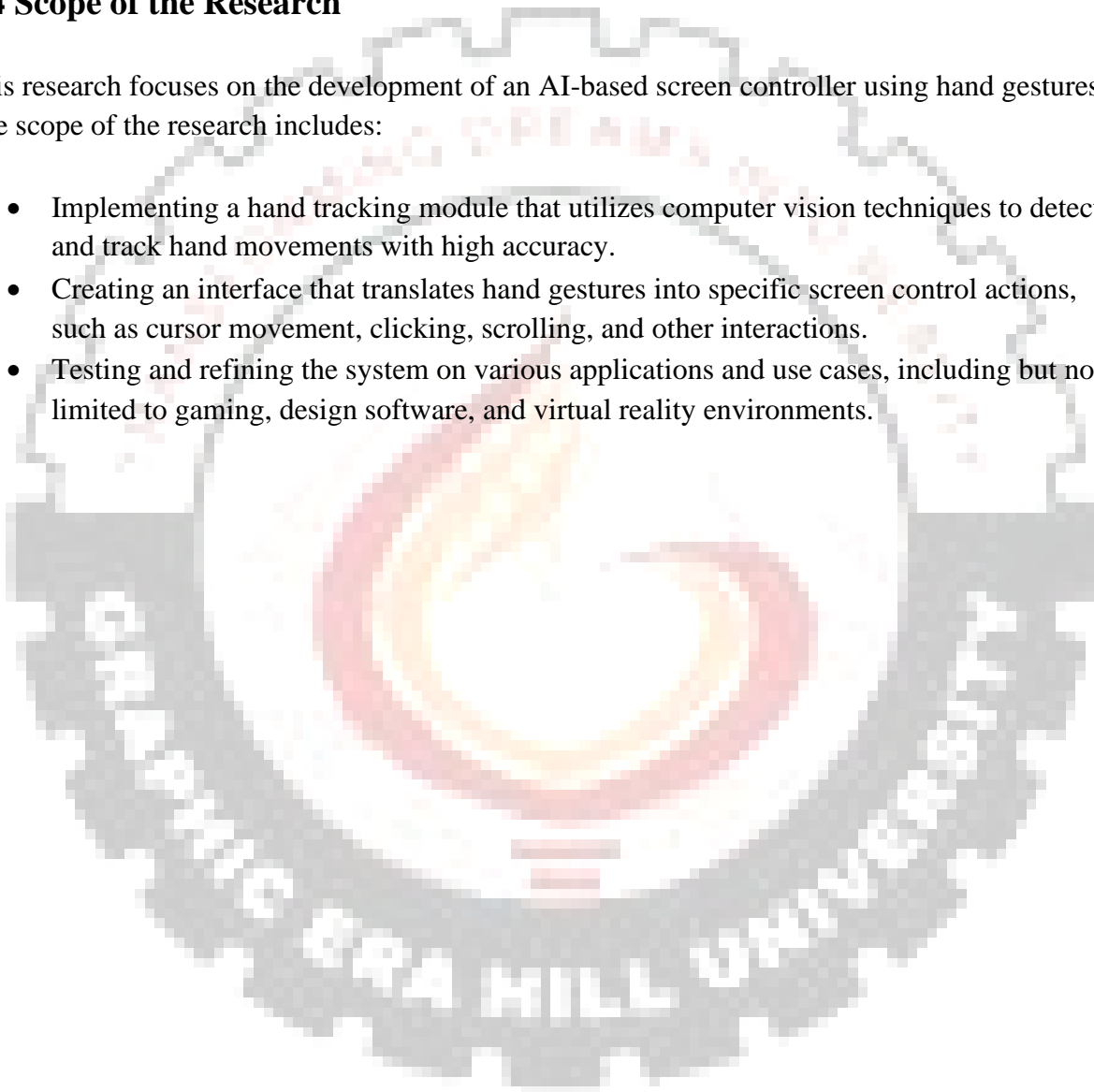
- Designing and implementing a hand tracking module that accurately detects and tracks hand movements in real-time.
- Developing algorithms to interpret hand gestures and map them to specific screen control actions.

- Integrating the screen controller with the operating system to enable seamless interaction with various applications and functionalities.
- Evaluating the performance and usability of the system through user studies and comparing it with traditional input devices.

## 1.4 Scope of the Research

This research focuses on the development of an AI-based screen controller using hand gestures. The scope of the research includes:

- Implementing a hand tracking module that utilizes computer vision techniques to detect and track hand movements with high accuracy.
- Creating an interface that translates hand gestures into specific screen control actions, such as cursor movement, clicking, scrolling, and other interactions.
- Testing and refining the system on various applications and use cases, including but not limited to gaming, design software, and virtual reality environments.





# Literature Review

## 2.1 Gesture Recognition Techniques

Gesture recognition techniques play a crucial role in the field of human-computer interaction, enabling users to interact with computers and devices using hand gestures. There are various techniques employed for gesture recognition, including computer vision-based and machine learning-based approaches.

### 2.1.1 Computer Vision-based Gesture Recognition

Computer vision-based gesture recognition techniques utilize image processing and computer vision algorithms to detect and interpret hand gestures. These techniques typically involve extracting hand features, such as hand shape, position, and movement, from images or video frames. In the presented project, the Hand Tracking module employs computer vision techniques to detect and track hands in real-time using the Media pipe library. It utilizes landmark detection to identify the positions of specific hand landmarks, allowing for gesture recognition based on hand poses and finger configurations.

### 2.1.2 Machine Learning-based Gesture Recognition

Machine learning-based gesture recognition techniques leverage the power of machine learning algorithms to recognize and classify hand gestures. These techniques involve training models on large datasets of hand gesture examples to learn patterns and relationships between input gestures and their corresponding outputs. The trained models can then be used to recognize and interpret new gestures. While the presented project focuses on computer vision-based techniques, machine learning algorithms can be integrated to enhance gesture recognition capabilities by training models to recognize specific hand gestures with high accuracy.

## 2.2 AI-based Human-Computer Interaction

AI-based human-computer interaction refers to the application of artificial intelligence techniques in enhancing the interaction between humans and computers. In the context of the project, the integration of hand tracking and gesture recognition with an AI virtual mouse allows users to control the mouse cursor on the screen using hand movements and gestures. This AI-powered interaction enables a more intuitive and natural way of interacting with computers, offering users greater flexibility and convenience.

## **2.3 Hand Tracking and Detection**

Hand tracking and detection form the foundation of the presented project. The Hand Tracking module utilizes computer vision techniques and the Mediapipe library to detect and track hands in real-time video frames. It extracts hand landmarks and calculates their positions, allowing for precise tracking and analysis of hand movements and gestures. The module employs a hand detection model and hand landmark models provided by Media pipe, enabling accurate and robust hand tracking capabilities.

## **2.4 Hand Gesture Recognition Systems**

Hand gesture recognition systems are designed to interpret and recognize specific hand gestures performed by users. These systems utilize various techniques, such as computer vision, machine learning, or a combination of both, to analyze hand movements, hand configurations, and finger poses. The presented project implements hand gesture recognition by analyzing the positions of hand landmarks and the configuration of fingers. It recognizes gestures such as moving the mouse cursor, clicking, right-clicking, and taking screenshots based on the detected finger states and hand poses.

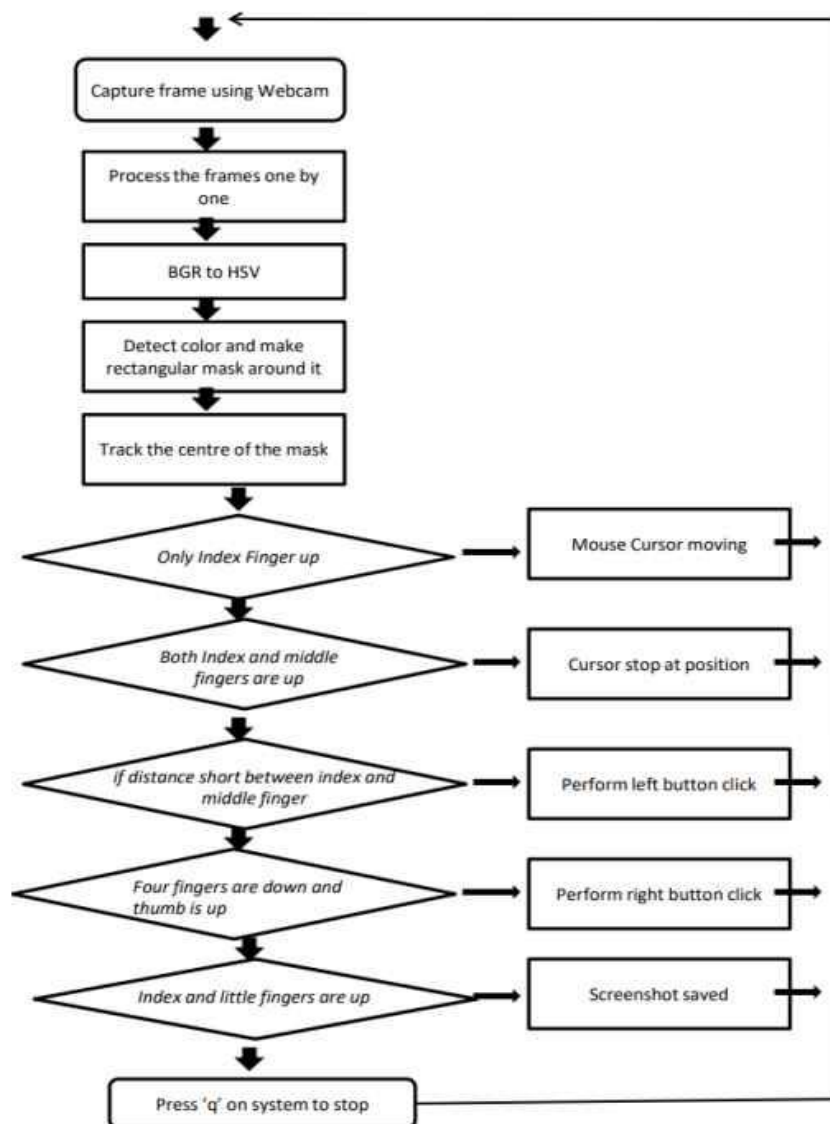
## **2.5 Summary**

In summary, this literature review explored gesture recognition techniques, focusing on computer vision-based and machine learning-based approaches. It discussed the role of AI in human computer interaction and the importance of hand tracking and detection in enabling gesture recognition. The review also highlighted the significance of hand gesture recognition systems in facilitating natural and intuitive user interactions with computers. The presented project leverages computer vision techniques, hand tracking, and gesture recognition to create an AI virtual mouse, enabling users to control the mouse cursor using hand gestures.

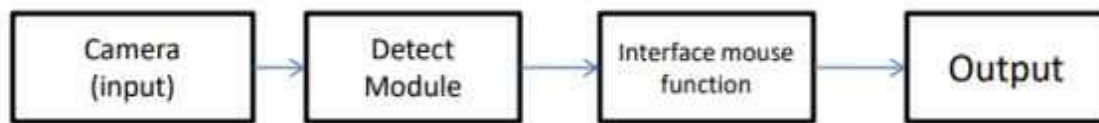
# System Architecture and Methodology

## 3.1 System Overview

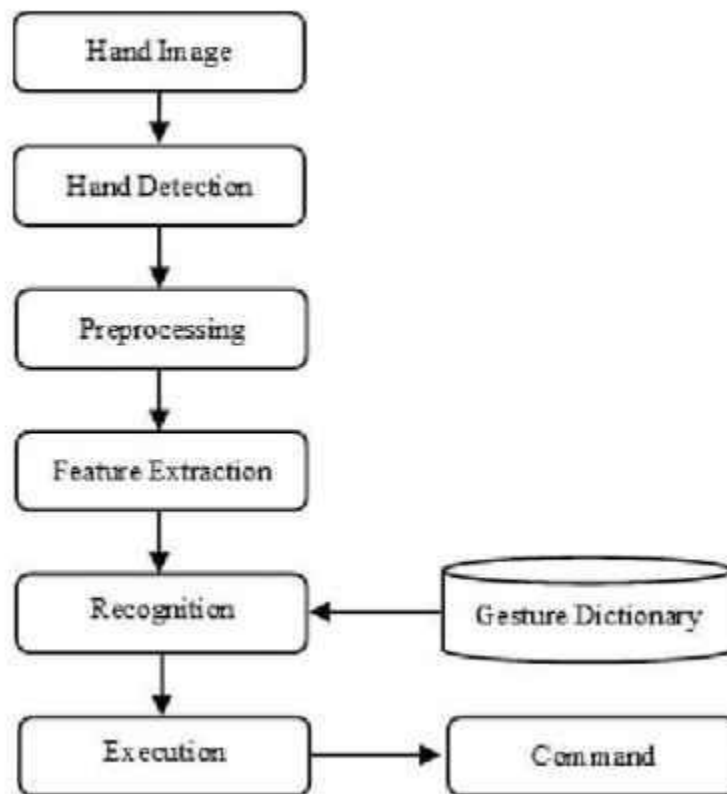
The AI-based screen controller using hand gestures is designed to enable intuitive screen control through the recognition of hand movements and gestures. The system comprises several key components, including data collection and preprocessing, hand tracking module, gesture recognition module, and AI-based screen control algorithms. These components work together to process input hand gestures and perform corresponding screen control actions.



### 3.2 Block Diagram of system



### 3.3 Flow chart of Modules of System



## System Development & Implementation

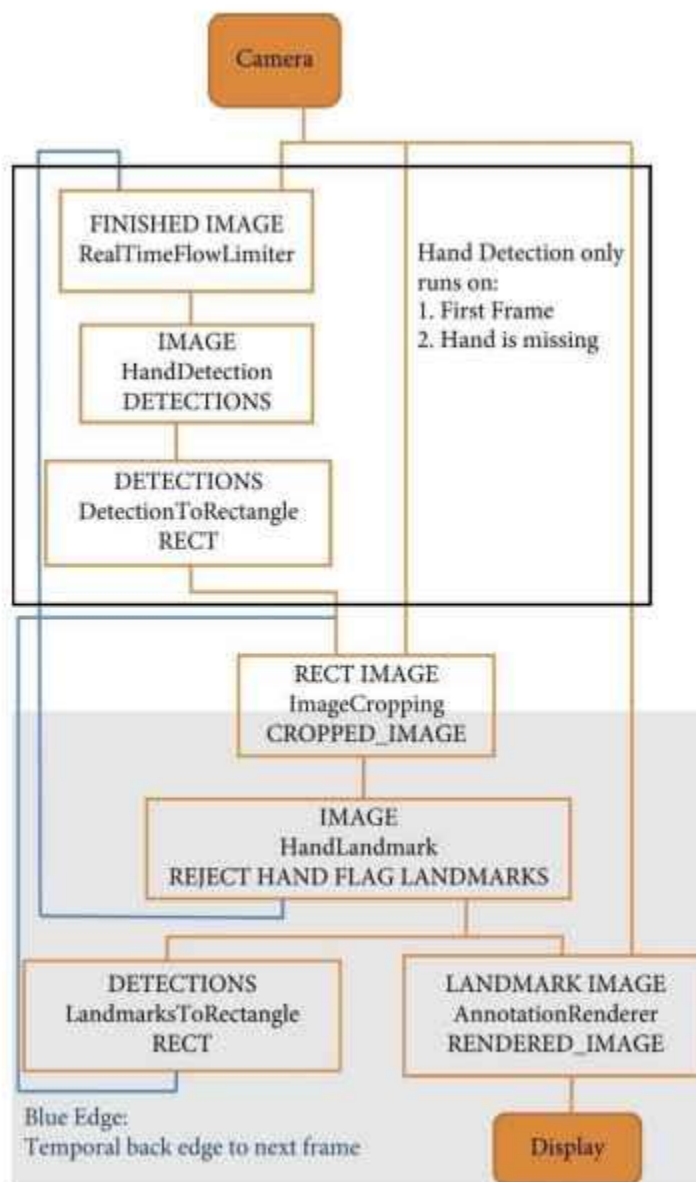
### 4.1 Tools and Language

- **Python 3.7** : Python programming language was the primary language used for developing the hand tracking mouse control system. Python offers simplicity, readability, and a vast array of libraries that make it suitable for computer vision and machine learning tasks.
- **PyCharm**: PyCharm is an integrated development environment (IDE) specifically designed for Python development. It provides features like code editing, debugging, and version control, which streamline the development process.
- **OpenCV**: OpenCV (Open Source Computer Vision Library) is a popular open-source computer vision library used for image and video processing tasks. It provides various functions and algorithms for handling images, videos, and webcam inputs. In this project, OpenCV was used for video capture, image processing, and rendering.
- **Mediapipe**: Mediapipe is a framework developed by Google that provides a comprehensive set of tools and algorithms for building applications involving real-time perception. The Hand Tracking Module used in your project is part of the Mediapipe library. It offers pretrained models for hand detection and tracking, making it easy to extract hand landmarks and gestures.
- **Autopy**: Autopy is a cross-platform library that allows interaction with the mouse, keyboard, and screen. It provides functionalities to control mouse movements, clicks, and keyboard inputs. In your project, Autopy was utilized for moving the mouse cursor and performing mouse clicks based on hand gestures.
- **Numpy**: Numpy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. Numpy was used in your project for various mathematical calculations and array manipulations. g. pytsx3: pytsx3 is a Python library that provides a simple interface for text-to-speech synthesis. It enables the system to speak out messages or notifications. In your project, pytsx3 was employed to generate audio notifications when capturing screenshots or performing specific hand gestures.

### 4.1.1 Libraries and their Purposes:

♣ **OpenCV:** Used for video capture, image processing, and rendering. It provides essential functions for handling image data, such as color space conversions, filtering, and drawing.

♣ **Mediapipe:** Integrated the Hand Tracking Module from Mediapipe, which offers pretrained models for hand detection and tracking. It allows the extraction of hand landmarks and the ability to recognize hand gestures.



♣ **Autopy:** Utilized Autopy library to control mouse movements and perform mouse clicks based on hand gestures. It provides an interface to interact with the mouse, keyboard, and screen.

♣ **Numpy:** Utilized Numpy for mathematical calculations, array manipulation, and handling multi-dimensional arrays. It simplifies complex mathematical operations in Python.

♣ **pyttsx3:** Incorporated pyttsx3 library for text-to-speech synthesis, allowing the system to generate audio notifications or messages. It converts text-based messages into audible speech.

The combination of these tools, languages, and libraries enabled the development of an efficient hand tracking mouse control system. The integration of OpenCV and Mediapipe provided robust hand detection and tracking capabilities, while Autopy facilitated precise mouse control based on hand gestures. Numpy was instrumental in performing mathematical calculations, and pyttsx3 added audio notifications to enhance the user experience

## 4.2 Hand Tracking Module

### 4.2.1 Code Overview

The Hand Tracking module is a crucial component of the project that enables the detection, tracking, and analysis of hand movements. It consists of a class called hand Detector that encapsulates the functionality related to hand detection and tracking. The code is implemented using the OpenCV and Mediapipe libraries. The handDetector class is initialized with parameters such as mode, maxHands, detectionCon, and trackCon, which control the behavior of the hand tracking algorithm. The mpHands and mpDraw objects are used from the Mediapipe library for hand tracking and landmark visualization, respectively. Additionally, the tipIds list is defined to identify specific landmarks on the hand.

### 4.2.2 Hand Detection and Tracking

In the find Hands method of the hand Detector class, the input image is processed to detect and track hands using the hands .process function. The processed results are then used to draw landmarks on the image if the hands are detected. The draw\_landmarks function from mpDraw is employed for this purpose. The method returns the image with the drawn



landmarks.

### **4.2.3 Landmark Detection and Visualization**

The `findPosition` method in the `handDetector` class is responsible for extracting the positions of landmarks on the hand. It takes the processed image as input and retrieves the landmarks' coordinates by iterating through the landmark attribute of the detected hand. The x and y coordinates are normalized and converted to pixel values based on the image dimensions. These coordinates are stored in the `lmList` list. Additionally, the method calculates the bounding box (`bbox`) that encompasses the hand by finding the minimum and maximum x and y values from the landmark coordinates. The bounding box is visualized on the image using the `rectangle` function from OpenCV. The method returns the `lmList` and the bounding box coordinates.

### **4.2.4 Finger Tracking and Gesture Recognition**

The `fingersUp` method in the `handDetector` class determines which fingers are extended based on the landmark positions. It compares the y-coordinate of specific landmarks (stored in the `tipIds` list) to detect whether the corresponding fingers are up or down. The method returns a list of binary values indicating the state of each finger.

### **4.2.5 Distance Calculation**

The `findDistance` method in the `handDetector` class calculates the Euclidean distance between two specified landmarks on the hand. It takes the indices of the two landmarks, the image, and optional parameters for visualization (radius and line thickness). The method retrieves the coordinates of the landmarks from the `lmList` and calculates the distance using the `hypot` function from the `math` module. It also draws a line and circles representing the landmarks on the image if the `draw` parameter is set to `true`. The method returns the distance, the modified image, and the coordinates of the line and circles. The Hand Tracking module provides essential functionality for the project, including hand detection, landmark visualization, finger tracking, and distance calculation. These features lay the foundation for the subsequent AI Virtual Mouse module, which utilizes the hand tracking capabilities to enable mouse control through hand gestures.

## **4.3 Ai Virtual Mouse Coding**

In the Ai Virtual Mouse section of the project, the code enables controlling the mouse cursor on the screen using hand gestures. It utilizes the Hand Tracking module to detect and track hand movements in real-time. The following parts describe the functionality and implementation of different features in the code.



### 4.3.1 Hand Tracking Initialization

The Hand Tracking module is initialized, setting the maximum number of hands to track, detection and tracking confidence levels, and the Hand Tracking Module object for hand detection and tracking.

- wCam and hCam: Width and height of the video frame from the camera.
- frameR: Frame reduction value for defining the region of interest (ROI).
- smoothening: Smoothening factor for stabilizing cursor movement.

### 4.3.2 Hand Tracking and Gesture Recognition Loop

The main loop continuously reads frames from the video capture, performs hand tracking, detects hand gestures, and interacts with the mouse accordingly.

- The loop runs indefinitely until interrupted.
- Within the loop, it reads frames from the video capture and passes them to the Hand Tracking module for detecting and tracking hands.
- The findHands function draws landmarks on the detected hands in the image.

### 4.3.3 Moving the Mouse Cursor

- This part checks if the index finger is up and the middle finger is down, indicating moving mode.
- It converts the hand coordinates to screen coordinates using interpolation.
- The cursor movement is smoothened to provide a smoother user experience.
- The `autopy.mouse.move` function moves the mouse cursor based on the calculated coordinates.

### 4.3.4 Clicking the Mouse

- This part checks if both the index and middle fingers are up, indicating clicking mode.
- It calculates the distance between the fingertips using the `findDistance` function from the Hand Tracking module.
- If the distance is short (less than 40 pixels), it simulates a mouse click using the

autopy.mouse.click function.

#### **4.3.5 Right-Clicking**

- This part checks if all the fingers except the thumb are down and the thumb is up, indicating right-clicking mode.
- It simulates a right-click by pressing and releasing the right mouse button using the autopy.mouse.toggle function.

#### **4.3.6 Taking Screenshots**

- This part checks if the index and little fingers are up, indicating screenshot mode.
- It waits for half a second to avoid accidental triggering of the screenshot action.
- The code captures a screenshot using the autopy.bitmap.capture\_screen function and saves it with a timestamp as the filename.
- Additionally, it utilizes the text-to-speech engine (pyttsx3) to speak a message confirming that the screenshot has been saved.

#### **4.3.7 Hand Presence and Closed Fist Detection**

- This part updates the hand\_closed variable based on whether all fingers are down, indicating a closed fist.
- If the hand is not present, the hand\_closed variable is reset to False.
- The image with hand landmarks and visualizations is displayed using cv2.imshow.
- The loop continues until the 'q' key is pressed.
- After the loop, the video capture is released and all OpenCV windows are closed.

## Source Code

```
import cv2 as cv
import mediapipe as mp
from pynput.keyboard import Key, Controller

# Initialize Keyboard Controller
keyboard = Controller()

mp_draw = mp.solutions.drawing_utils # Function to Draw Landmarks over Hand
mp_hand = mp.solutions.hands # Hand Detection Function

fingerTipIds = [4, 8, 12, 16, 20]

# Capturing the Video from the Camera
video = cv.VideoCapture(0)

# Initializing the Hand Detection Function
hands = mp_hand.Hands(min_detection_confidence = 0.5, min_tracking_confidence = 0.5)

while True:
    success, image = video.read()
```

```
# Converting the Image to RGB
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

# Processing the Image for Hand Detection
image.flags.writeable = False
results = hands.process(image)
image.flags.writeable = True

# Converting the Image back to BGR
image = cv.cvtColor(image, cv.COLOR_RGB2BGR)

# List to store the Landmark's Coordinates
landmarks_list = []

# If Landmarks Detected i.e., Hand Detected Successfully
if results.multi_hand_landmarks:
    hand_landmarks = results.multi_hand_landmarks[-1]

    for index, lm in enumerate(hand_landmarks.landmark):
        h, w, c = image.shape # Height, Width, Channels
        cx, cy = int(lm.x*w), int(lm.y*h)
        landmarks_list.append([index, cx, cy])

# Drawing the Landmarks for only One Hand
# Landmarks will be drawn for the Hand which was Detected First
```

```
mp_draw.draw_landmarks(image, hand_landmarks,  
mp_hand.HAND_CONNECTIONS)
```

```
# Stores 1 if finger is Open and 0 if finger is closed
```

```
fingers_open = []
```

```
if len(landmarks_list) != 0:
```

```
    for tipId in fingerTipIds:
```

```
        if tipId == 4: # That is the thumb
```

```
            if landmarks_list[tipId][1] > landmarks_list[tipId - 1][1]:
```

```
                fingers_open.append(1)
```

```
            else:
```

```
                fingers_open.append(0)
```

```
        else:
```

```
            if landmarks_list[tipId][2] < landmarks_list[tipId - 2][2]:
```

```
                fingers_open.append(1)
```

```
            else:
```

```
                fingers_open.append(0)
```

```
# Counts the Number of Fingers Open
```

```
count_fingers_open = fingers_open.count(1)
```

```
# If Hand Detected
```

```
if results.multi_hand_landmarks != None:
```

```
    # If All Fingers Closed --> Break
```

```
if count_fingers_open == 0:

    cv.rectangle(image, (20, 300), (270, 425), (0, 255, 0), cv.FILLED)

    cv.putText(image, "BRAKE", (45, 375), cv.FONT_HERSHEY_SIMPLEX, 2,
(255, 0, 0), 5)

    keyboard.press(Key.left)
    keyboard.release(Key.right)

# If All Fingers Open --> Gas
elif count_fingers_open == 5:

    cv.rectangle(image, (20, 300), (270, 425), (0, 255, 0), cv.FILLED)

    cv.putText(image, "GAS", (45, 375), cv.FONT_HERSHEY_SIMPLEX, 2, (255, 0,
0), 5)

    keyboard.press(Key.right)
    keyboard.release(Key.left)

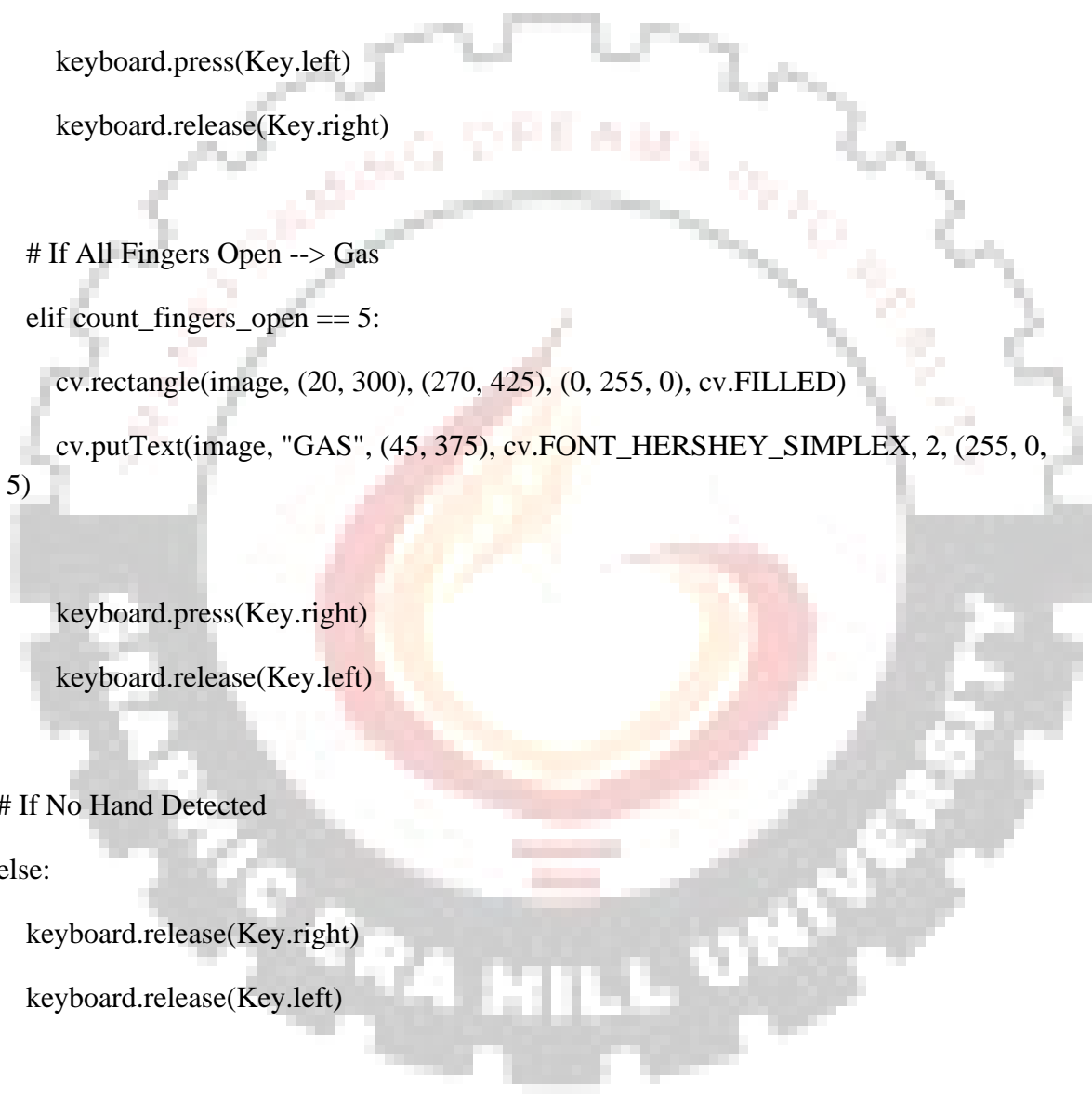
# If No Hand Detected
else:

    keyboard.release(Key.right)
    keyboard.release(Key.left)

# Show the Video

cv.imshow("Frame", image)

# Close the Video if "q" key is pressed
```



```
if cv.waitKey(1) & 0xFF == ord('q'):  
    break  
  
video.release()  
cv.destroyAllWindows()
```



## Conclusion and Future Work

### 6.1 Limitations and Challenges:

While the proposed system offers a promising solution for hand gesture recognition and AI-based human-computer interaction, it is important to acknowledge the limitations and challenges that need to be addressed:

- **Occlusion issues:** One of the primary challenges faced by the system is handling occlusion. When the hand is partially or fully occluded, such as when fingers overlap or objects obstruct the hand, it can lead to inaccurate hand tracking and gesture recognition. To overcome this limitation, advanced techniques such as 3D hand tracking or multi-camera setups can be explored to improve the system's robustness in occlusion scenarios.
- **Lighting conditions:** Variations in lighting conditions can significantly impact the system's performance. Low light conditions can lead to difficulties in detecting hand landmarks accurately, while strong backlighting can result in overexposed images, making hand tracking challenging. To address this, adaptive algorithms that dynamically adjust the image processing parameters based on the lighting conditions can be implemented. Additionally, exploring techniques like infrared-based hand tracking can provide more reliable results in diverse lighting environments.
- **Complex gestures:** Recognizing complex gestures involving multiple fingers or intricate hand movements can be a challenging task. Certain gestures may have similar hand configurations, making it difficult to distinguish them accurately. To overcome this, incorporating advanced machine learning algorithms, such as deep learning models, can help capture more complex patterns and improve gesture recognition accuracy. Additionally, integrating temporal information by considering the sequence of hand poses over time can enhance the system's ability to recognize dynamic gestures.



## 6.2 Future Enhancements:

To further advance the proposed system and address its limitations, the following future enhancements can be considered:

- **Advanced machine learning algorithms:** Expanding the system's capabilities by exploring and integrating advanced machine learning techniques, such as deep learning models, can significantly improve the accuracy and robustness of gesture recognition. These models can learn complex patterns and representations from large-scale training data, enabling the system to recognize a wider range of gestures accurately.
- **Refinement of hand tracking module:** Continuously refining the hand tracking module is crucial for achieving more accurate and reliable hand tracking. Improvements can be made to handle occlusion and lighting challenges by exploring novel algorithms that leverage multiple sensors or depth information. Additionally, incorporating hand pose estimation techniques can provide more detailed information about hand articulation, enabling precise tracking even in challenging conditions.
- **User customization and adaptation:** Providing options for users to calibrate the system according to their hand shape and size can enhance the user experience and improve gesture recognition accuracy. By allowing users to input their hand profiles or leveraging user-specific training, the system can adapt to individual variations, ensuring personalized interaction.
- **Multi-user support:** Extending the system's capabilities to support multiple users simultaneously can enable collaborative interactions and expand its applicability in scenarios such as interactive presentations or group activities. Techniques like multi-camera setups or advanced depth sensing can be explored to differentiate between multiple users' hands and enable simultaneous tracking and interaction.
- **Integration of additional features:** To enhance the overall usability and flexibility of

the system, integrating additional features can be considered. For example, incorporating voice commands alongside hand gestures can provide users with multiple interaction modalities.

- **Cross-platform compatibility:** Expanding the system's compatibility to various platforms and operating systems can increase its accessibility and reach. Supporting different devices, such as smartphones, tablets, or wearable devices, can enable users to interact with a wide range of digital interfaces beyond traditional desktop environments.

This cross-platform compatibility can be achieved through platform-specific adaptations and optimizations. By addressing these limitations and exploring the suggested future enhancements, the proposed system can further improve its accuracy, usability, and adaptability, making it a more powerful tool for AI-based human-computer interaction.



## Bibliography

D.-H. Liou, D. Lee, and C.-C. Hsieh, "A real time hand gesture recognition system using motion history image," in Proceedings of the 2010 2nd International Conference on Signal Processing Systems, IEEE, Dalian, China, July 2010.

L. Thomas, "Virtual mouse using hand gesture," International Research Journal of Engineering and Technology (IRJET), vol. 5, no. 4, 2018.

S. U. Dudhane, "Cursor control system using hand gesture recognition," IJARCCCE, vol. 2, no. 5, 2013.

D. L. Quam, "Gesture recognition with a DataGlove," IEEE Conference on Aerospace and Electronics, vol. 2, pp. 755–760, 1990.

Katona, "A review of human–computer interaction and virtual reality research fields in cognitive InfoCommunications," Applied Sciences, vol. 11, no. 6, p. 2646, 2021.