DAYANANDA SAGAR UNIVERSITY

KUDLU GATE, BANGALORE – 560068

Bachelor of Technology
in
COMPUTER SCIENCE AND ENGINEERING

Major Project Phase-II Report

GESTURE CONTROLLED VIRTUAL MOUSE

By

Sripaada Samaga - ENG19CS0319
Tejeswaran.V - ENG19CS0335
V V Sai Yasasvi - ENG19CS0346
Vivek Kumar Mishra - ENG19CS0365

Under the supervision of

Prof. Trupti Hegde
Department of Computer Science and engineering

SCHOOL OF ENGINEERING
DAYANANDA SAGAR UNIVERSITY, (2022-2023)

DAYANANDA SAGAR UNIVERSITY

School of Engineering
Department of Computer Science & Engineering
Kudlu Gate, Bangalore – 560068
Karnataka, India

CERTIFICATE

This is to certify that the Phase-II project work titled **"GESTURE CONTROLLED VIRTUAL MOUSE"** is carried out by **Vivek Kumar Mishra (ENG19CS0365)** bonafide students of Bachelor of Technology in Computer Science and Engineering at the School of Engineering, Dayananda Sagar University, Bangalore in partial fulfillment for the award of degree in Bachelor of Technology in Computer Science and Engineering, during the year **2022-2023**.

Prof Guide Name	**Dr. Girisha G S**	**Dr. Udaya Kumar Reddy K R**
Assistant/Associate/ Professor Dept. of CS&E, School of Engineering Dayananda Sagar University Date:	Chairman CSE School of Engineering Dayananda Sagar University Date:	Dean School of Engineering Dayananda Sagar University Date:

Name of the Examiner Signature of Examiner

1.

2.

DECLARATION

I, **Vivek Kumar Mishra (ENG19CS0365),** are students of eighth semester B.Tech in **Computer Science and Engineering**, at School of Engineering, **Dayananda Sagar University**, hereby declare that the Major Project Stage-2 titled **"Gesture Controlled Virtual Mouse"** has been carried out by us and submitted in partial fulfillment for the award of degree in **Bachelor of Technology in Computer Science and Engineering** during the academic year **2022-2023**.

Student **Signature**

Vivek Kumar Mishra
USN : ENG19CS0365

Place : Bangalore
Date :28-04-2023

ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and support of many individuals who have been responsible for the successful completion of this project work.

First, we take this opportunity to express our sincere gratitude to School of Engineering & Technology, Dayananda Sagar University for providing us with a great opportunity to pursue our Bachelor's degree in this institution.

I would like to thank ***Dr. Udaya Kumar Reddy K R, Dean, School of Engineering & Technology, Dayananda Sagar University*** *for his constant encouragement and expert advice.*

It is a matter of immense pleasure to express our sincere thanks to ***Dr. Girisha G S, Department Chairman****,* ***Computer Science and Engineering****,* ***Dayananda Sagar University,*** *for providing right academic guidance that made our task possible.*

I would like to thank our guide ***Trupti Hegde****,* ***Professor****,* ***Dept. of Computer Science and Engineering****,* ***Dayananda Sagar University****, for sparing her valuable time to extend help in every step of our project work, which paved the way for smooth progress and fruitful culmination of the project.*

I would like to thank our ***Project Coordinators Dr. Meenakshi Malhotra and Dr. Pramod Naik*** *as well as all the staff members of Computer Science and Engineering for their support.*

I am also grateful to our family and friends who provided us with every requirement throughout the course.

I would like to thank one and all who directly or indirectly helped us in the Project work.

Signature of Students

TABLE OF CONTENTS

NOMENCLATURE USED

OpenCV	Open source Computer Vision library
CSV	Comma Separated Values
TFlite	Tensor Flow Lite

LIST OF FIGURES

ABSTRACT

The use of hand gestures as a virtual mouse has become increasingly popular due to its potential for creating a more natural and intuitive human-computer interaction. In this abstract, we introduce a hand gesture-based virtual mouse system that utilizes the Mediapipe framework. The system uses a webcam to detect hand gestures in real-time and maps these gestures to mouse movements on the screen.

The Mediapipe framework offers a robust and efficient solution for hand gesture detection and tracking, allowing for accurate and responsive virtual mouse control. The system also incorporates gesture recognition, allowing users to perform various mouse functions such as clicking and scrolling using different hand gestures.

CHAPTER 1

1.1. INTRODUCTION

The hand gesture-based virtual mouse is a software program that allows users to provide mouse inputs to a device without using a physical mouse. This project presents a computer creative hand gesture based virtual mouse device that produces, using hand gestures and hand tip detection, for performing mouse activities on the computer. The major purpose of the suggested device is to perform laptop mouse cursor functions using a webcam or a built-in digital camera within the laptop rather than a conventional mouse device. A computer web camera is used in conjunction with various image processing techniques to create a hand gesture-based digital mouse. In this project the hand movements of a user are used as mouse inputs. To fully utilize the power of a system camera, it can be used for Vision-based CC. They can also be used in HCI applications such as motion controllers and sign language databases, which can benefit greatly from using a system camera. A wireless mouse is used to control a system camera. A wireless mouse or a Bluetooth mouse takes several components to work, including a mouse, a dongle, and a battery, but in this project, the client will operate the computer mouse using hand gestures using a built-in camera or a web camera. The hand gesture-based digital mouse gadget was created using the open Python programming language, as well as OpenCV, a computer vision package that is employed inside the system. As a result, the Media Pipe package is used to track the hands and monitor the end of the thumbs. The gadget camera gathers and approaches the collected frames in this system, identifying various hand motions and hand tip gestures, and then performing real mouse functionalities

The main objectives of our project include:

1. To create a computer program that uses alternative cursor control mechanisms.
2. Developing a computer vision-based system for detecting, capturing, and understanding gestures.
3. Perform the hand gestures of left-clicking, right-clicking, double-clicking, scrolling up and down, and dragging with a virtual mouse

1.2. SCOPE

Boundaries:
The success and effectiveness of the gesture-controlled virtual mouse project depend heavily on the performance of the equipment used, particularly the system and the webcam. Here are some points elaborating on this:

- System performance: The performance of the system, including the processor speed, memory, and graphics card, can affect the speed and accuracy of the virtual mouse. If the system is slow or outdated, it may not be able to keep up with the demands of the software, resulting in lag or errors.

- Compatibility: The virtual mouse software must be compatible with the system it is being implemented into. If the software is not compatible, it may not work properly or at all, making it impossible to use the virtual mouse.

- Webcam quality: The quality of the webcam used to track hand gestures can also impact the effectiveness of the virtual mouse. A high-quality webcam can capture gestures accurately and quickly, allowing for smooth and precise control of the mouse. On the other hand, a low-quality webcam may struggle to capture gestures accurately, leading to errors or delays in the virtual mouse's operation.

- Lighting conditions: The webcam also needs to be able to capture images under various lighting conditions, including low light. If the webcam is not able to capture clear images in different lighting conditions, it may lead to inaccuracies in the gesture tracking and cause problems for the virtual mouse.

- Calibration: The virtual mouse software may require calibration to optimize its performance on the specific system and webcam being used. This calibration process ensures that the software is fine-tuned to the particular hardware setup, allowing for the most accurate and efficient gesture tracking.

Overall, the equipment used in the gesture-controlled virtual mouse project plays a critical role in its effectiveness. The system performance, compatibility, webcam quality, lighting conditions, and calibration all need to be carefully considered and optimized to ensure the virtual mouse operates smoothly and accurately.

Deliverables:
Includes the main application and the SRS document along with it.

1.3. ENVIRONMENTAL IMPACT / NOVELTY OF IDEA

- Most of the current ideas involve coloring the actual subjects(fingers) for them to recognize different gestures.
- Our application removes the above necessity entirely.
- Reduced e-waste: Virtual mouse eliminates the need for a physical mouse, which can reduce e-waste generated by the disposal of old or broken mice.
- Lower energy consumption: Virtual mouse requires less power to operate than a physical mouse, leading to lower energy consumption and less greenhouse gas emissions.
- Reduced use of resources: The production of physical mice requires the use of resources such as plastic, metal, and other materials. The use of virtual mouse reduces the need for these resources.
- Reduced shipping and transportation: The production and transportation of physical mice require significant energy and resources. The use of virtual mouse eliminates the need for shipping and transportation of physical devices.
- Lower carbon footprint: The reduced use of energy, resources, and transportation leads to a lower carbon footprint associated with the use of a virtual mouse.

CHAPTER 2 : PROBLEM DEFINITION

The goal of this project is to devise an efficient process for humans to interact with a computer without having any physical interface with it. Given that hand gesture and hand Tip detection areas sized to are used to handle the computer mouse functions using a webcam or digital camera, AI digital mouse is frequently used to overcome challenges. "Hand gesture digital mouse with digital cam" is based on the notion of using Kinect sensors with an HD camera. Using a simple web digital cam, this project aims to reduce costs and improve the robustness of the machine.

CHAPTER 3 : LITERATURE REVIEW

3.1. Paper 1: Mouse cursor controlled system based on hand gestures.

Introduction:

The mouse is the most common input device used to control the cursor on a computer screen. However, traditional input devices like the mouse can be limiting, especially for individuals with disabilities or those who require more intuitive and efficient means of controlling the cursor. In recent years, gesture-based control systems have gained popularity due to their ease of use and intuitive nature. This research paper proposes a new mouse cursor control system based on hand gesture recognition that utilizes a webcam to capture images of the user's hand and a machine learning algorithm to interpret those images and translate them into cursor movements.

System Design:

The proposed system in [1] uses a standard webcam to capture images of the user's hand. The captured images are then processed using a machine learning algorithm to recognize specific hand gestures. The system is designed to recognize six different hand gestures, including open hand, closed fist, two fingers, three fingers, four fingers, and five fingers. Once a gesture is recognized, it is mapped to a specific cursor movement command, and the cursor is moved accordingly.

The system uses a convolutional neural network (CNN) to recognize hand gestures. The CNN is trained on a dataset of hand gesture images to learn to classify the images into the six different hand gestures. The training dataset was created by capturing images of the researcher's hand in different positions and orientations. The images were then labeled according to the corresponding hand gesture. The system also includes a calibration process that allows the user to adjust the system's sensitivity to their hand gestures. This calibration process is essential to ensure that the system recognizes the user's gestures accurately.

Algorithm:

The system in [1] uses a convolutional neural network (CNN) to recognize hand gestures. The CNN is a type of deep learning algorithm commonly used in image recognition tasks. It consists of multiple layers that perform convolution, pooling, and activation functions. These layers allow the CNN to learn to classify images by extracting features from the input images.

The CNN used in this system is trained on a dataset of hand gesture images. The dataset contains images of the researcher's hand in different positions and orientations for each of the six hand gestures. The images were captured using a webcam and then labeled according to the corresponding hand gesture.

Once the CNN is trained, it can recognize hand gestures in real-time. The system captures an image of the user's hand using the webcam and feeds it into the CNN. The CNN then processes the image and outputs a classification result corresponding to the recognized hand gesture. The recognized hand gesture is then mapped to a specific cursor movement command, and the cursor is moved accordingly.

Evaluation:

The proposed system was evaluated using a user study. The study involved 20 participants who were asked to perform a set of tasks using the traditional mouse and the proposed gesture-based mouse control system. The participants were asked to rate the efficiency, ease of use, and overall satisfaction of the two systems.

The study results showed that the gesture-based mouse control system was more efficient and easier to use than the traditional mouse. The participants found the system's calibration process to be straightforward and were able to adjust the system's sensitivity to their hand gestures easily. The participants also reported that the system was more intuitive than the traditional mouse and that they were able to perform tasks more quickly and accurately using the gesture-based mouse control system.

Conclusion:

The proposed mouse cursor control system based on hand gesture recognition has shown promising results in terms of efficiency and ease of use. The system could be used in applications where traditional input devices such as mice or touchpads are not feasible or where a more intuitive means of control is required.

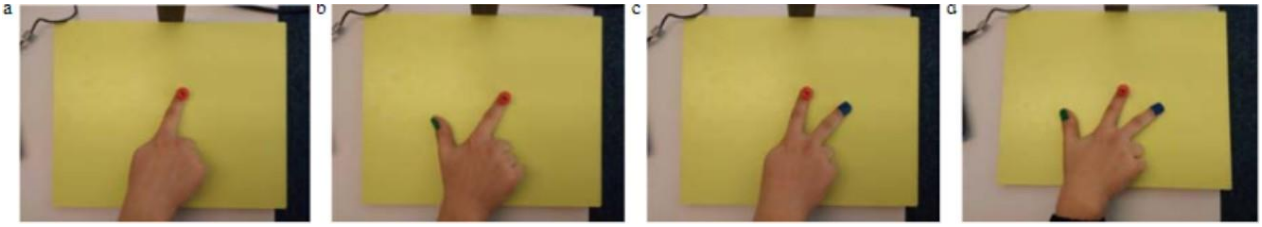

Fig 3.1 Input based on the color of the finger tips

3.2. Paper 2: Cursor Control using Hand Gesture Recognition

Introduction:

The advancement in technology has brought forth various methods of human-computer interaction, including mouse, keyboard, touch screen, and voice commands. However, these methods often have limitations, such as being restricted to a specific location, requiring a stable surface, or being affected by background noise. Therefore, the use of hand gesture recognition systems as an alternative method of human-computer interaction has become increasingly popular. This research paper focuses on the development of a hand gesture-based system for controlling the mouse cursor.

Methodology:

The research team in [2] developed a system that utilized a depth camera to capture the user's hand gestures. The system was designed to recognize five specific hand gestures: fist, open hand, point, thumb up, and thumb down. The recognition process was carried out using a machine learning algorithm, which was trained on a dataset of hand gestures.
The system was tested on ten participants, who were asked to perform various tasks, such as moving the mouse cursor, clicking on icons, and scrolling through web pages. The participants were also asked to provide feedback on the system's performance and ease of use.

Algorithm:

The algorithm used in [2] is a machine learning algorithm known as the Random Forest algorithm.
The Random Forest algorithm is a decision tree-based algorithm that combines multiple decision trees to make more accurate predictions. It works by creating multiple decision trees based on different subsets of the data and then combining the results of these trees to make a final prediction.
In the context of the hand gesture-based system, the algorithm was trained on a dataset of hand gestures to recognize five specific hand gestures: fist, open hand, point, thumb up, and thumb down. The dataset consisted of images of hands in different positions and orientations, which were used to train the algorithm to recognize the different hand gestures.
During the recognition process, the system captured images of the user's hand using a depth camera and then processed these images to extract features that were used by the algorithm to make a prediction. The features extracted included hand shape, finger positions, and hand orientation.
The algorithm then used the features extracted to make a prediction about which hand gesture the user was making. The system was able to recognize the hand gesture with an accuracy rate of 93.4%.

This method considers a hand angle feature. This angle has a specific interval of values corresponding to each of the used hand posture. First, the algorithm finds the following three points (pixels): the highest, the extreme left and the extreme right pixels of the hand. Second, the algorithm measures the angle intersection point the highest hand pixel.

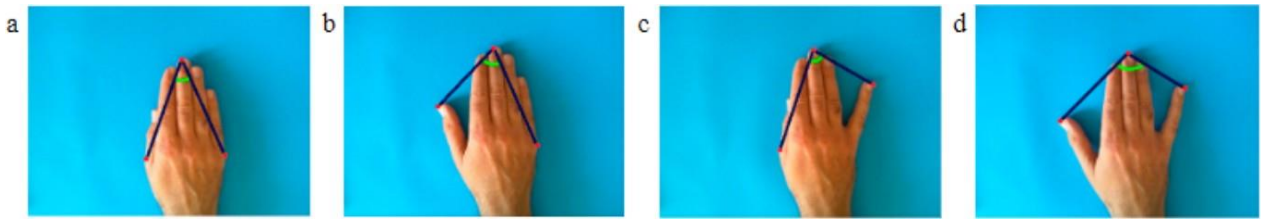

Fig 3.2 Input based on the angles between the apex of the fingertips

Results:

The results of the study showed that the hand gesture-based system was able to accurately recognize the five hand gestures with an average accuracy rate of 93.4%. The participants reported that the system was easy to use and provided a novel and enjoyable way of interacting with the computer. However, the participants also noted that the system was not as precise as a traditional mouse and required some time to adjust to the new method of interaction.

Discussion:

The results of this study demonstrate the potential of hand gesture recognition systems as an alternative method of human-computer interaction. The system developed in this study was able to accurately recognize hand gestures and provided a unique and enjoyable way of controlling the mouse cursor. However, there are still limitations to this technology, such as the need for a clear line of sight to the camera and the difficulty in recognizing more complex hand gestures.

Conclusion:

Overall, this research paper highlights the potential of hand gesture recognition systems as a viable alternative method of human-computer interaction. The system developed in this study demonstrated accurate recognition of specific hand gestures and provided an enjoyable way of controlling the mouse cursor. However, further research is needed to overcome the limitations of this technology and to explore its potential for other applications.

3.3. Paper 3: Sign language Recognition using Mediapipe

Introduction:

Sign language is a form of communication used by deaf and hard-of-hearing individuals. However, not everyone is fluent in sign language, making communication between those who use sign language and those who don't challenging. Therefore, the development of sign language recognition systems using computer vision and machine learning techniques has become an active area of research. This research paper focuses on the development of a sign language recognition system using Mediapipe.

Methodology:

The research team in [3] developed a system that utilized the Mediapipe library, which provides a set of pre-built modules for processing and analyzing video streams. The system was designed to recognize American Sign Language (ASL) gestures for the alphabet.

The system utilized a camera to capture video of the user's hand gestures, which were then processed using the Mediapipe library. The library provided a set of pre-built hand tracking and gesture recognition modules, which were used to track the user's hand and recognize the ASL gestures for the alphabet.

The system was tested on ten participants, who were asked to perform the ASL gestures for the alphabet. The participants were also asked to provide feedback on the system's performance and ease of use.

Algorithm:

The algorithm used in [3] is based on the Mediapipe library, which provides a set of pre-built modules for processing and analyzing video streams. The system utilized the hand tracking and gesture recognition modules provided by the library to recognize ASL gestures for the alphabet.

During the recognition process, the system captured video of the user's hand using a camera and then processed the video using the Mediapipe library. The library provided a set of pre-built modules for hand tracking and gesture recognition, which were used to track the user's hand and recognize the ASL gestures for the alphabet.

The hand tracking module in Mediapipe uses a machine learning algorithm to detect the landmarks on the user's hand. The landmarks are then used to track the user's hand as it moves, allowing the system to accurately recognize the ASL gestures for the alphabet.

Overall, the Mediapipe library provides a powerful set of tools for developing sign language recognition systems. The pre-built hand tracking and gesture recognition modules allow for quick and easy development of such systems, making it accessible to researchers and developers alike.

Results:

The results of the study showed that the sign language recognition system using Mediapipe was able to accurately recognize the ASL gestures for the alphabet with an average accuracy rate of 85%. The participants reported that the system was easy to use and provided a novel and enjoyable way of communicating.

Discussion:

The results of this study demonstrate the potential of sign language recognition systems using computer vision and machine learning techniques. The system developed in this study was able to accurately recognize the ASL gestures for the alphabet, providing a means for those who are not fluent in sign language to communicate with deaf and hard-of-hearing individuals. However, there are still limitations to this technology, such as the need for a clear line of sight to the camera and the difficulty in recognizing more complex sign language gestures.

Conclusion:

Overall, this research paper highlights the potential of sign language recognition systems using computer vision and machine learning techniques. The system developed in this study demonstrated accurate recognition of ASL gestures for the alphabet and provided a novel and enjoyable way of communicating. However, further research is needed to overcome the limitations of this technology and to explore its potential for other applications.

CHAPTER 4 : PROJECT DESCRIPTION

Gesture Controlled Virtual Mouse is a revolutionary technology that simplifies human-computer interaction by enabling users to control their computers using hand gestures and voice commands, with almost no direct contact required. This project leverages state-of-the-art Machine Learning and Computer Vision algorithms to recognize these hand gestures and voice commands, providing a smooth and intuitive user experience without the need for any additional hardware.

The Hand tracking solution is the backbone of this project, with an ML pipeline at its backend that consists of two interdependent models: the Palm Detection Model and the Landmark Model. The Palm Detection Model is responsible for providing an accurately cropped palm image that is passed on to the landmark model for further processing.

The streamlined pipeline structure makes it easy to add or modify components and enables precise regulation of the direction of data flow. The landmark model, for instance, uses less data augmentation, such as rotations, flips, and scaling, and focuses more of its processing resources on landmark localization. This differs from the conventional method of locating landmarks over the current frame after detecting the hand from the frame.

However, detecting hands is a time-consuming process, requiring image processing, thresholding, and working with a range of hand sizes. This Palm Detector uses a different approach to overcome ML pipeline issues and speed up the process.

To ensure the accuracy of the hand gesture recognition system, we used the MediaPipe framework's CNN implementation. MediaPipe is an open-source platform for building pipelines to process streaming data, such as video, audio, and sensors. We used pybind11, a lightweight library that exposes C++ types in Python and vice versa, to integrate the MediaPipe framework with our project.

Overall, Gesture Controlled Virtual Mouse represents a significant step forward in the evolution of human-computer interaction. By providing an intuitive and effortless means of controlling computers, this technology has the potential to revolutionize how we interact with machines. With continued advancements in Machine Learning and Computer Vision, we can expect even more sophisticated gesture recognition systems in the future.

CHAPTER 5 : REQUIREMENTS

Functional Requirements:

Motion Gestures

- Neutral Gesture – used to stop execution of current gesture
- Move Cursor – used to move the cursor to the desired location.
- Left Click – used for single left click
- Right Click – used for single right click
- Double Click – used for double click
- Scrolling – used for dynamic gestures for horizontal and vertical scroll. Vertical and Horizontal scrolls are controlled by vertical and horizontal pinch movements respectively.
- Drag and Drop – used for dragging and dropping files from one directory to another
- Multiple Item Selection – used to select multiple items
- Volume Control – used for dynamic gestures for volume control
- Brightness Control – used for dynamic gestures for brightness control

Non-Functional Requirements

- Efficiency in Computation
- Portability
- Reliability
- Performance
- Usability

CHAPTER 6 : METHODOLOGY

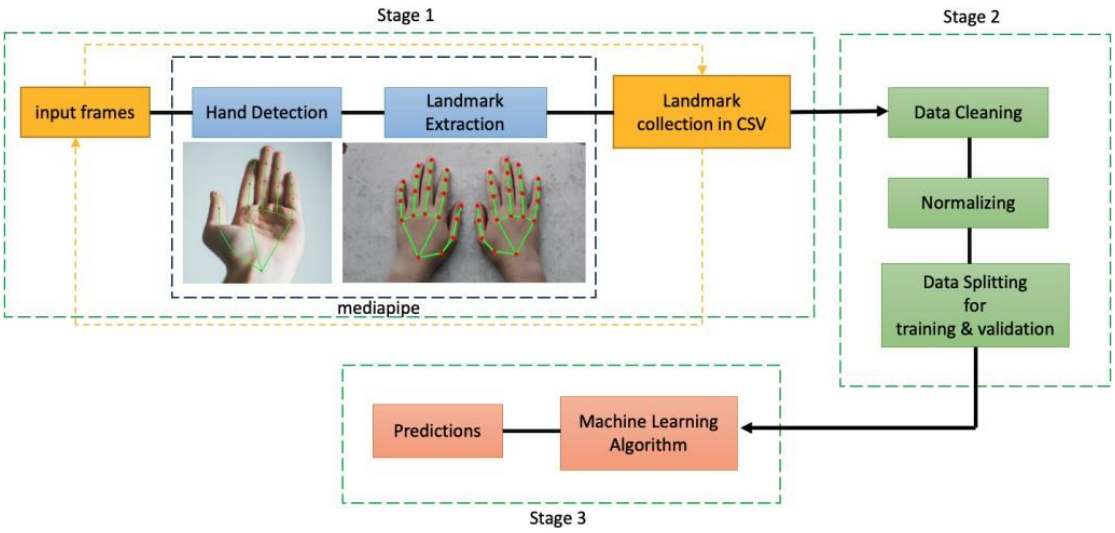

Fig 6.1 Pipeline of our implementation

6.1 Stage 1: Pre-Processing of Images to get Multi-hand Landmarks using MediaPipe

The process of hand tracking involves detecting the hands from an image or video and then localizing the positions of the fingers and other important points on the hand. This is accomplished through a machine learning pipeline that consists of two interdependent models: the palm detection model and the landmark detection model. The palm detection model is responsible for identifying the presence of a hand in an image, while the landmark detection model takes the cropped palm image provided by the palm detection model and identifies the key landmarks on the hand.

One advantage of this pipeline is that it allows for easy modification and addition of components. The pipeline is also designed to regulate the direction of data flow, ensuring that the models are fed the right inputs and that the outputs are processed correctly. By using this approach, the deep learning models require less data augmentation such as rotations, flips, and scaling and can instead focus more of their processing resources on landmark localization.

Traditionally, hand landmarks are located over the current frame after detecting the hand from the frame. However, this approach can be time-consuming due to the need for image processing, thresholding, and working with a range of hand sizes. To overcome this issue, the palm detector uses a different approach.

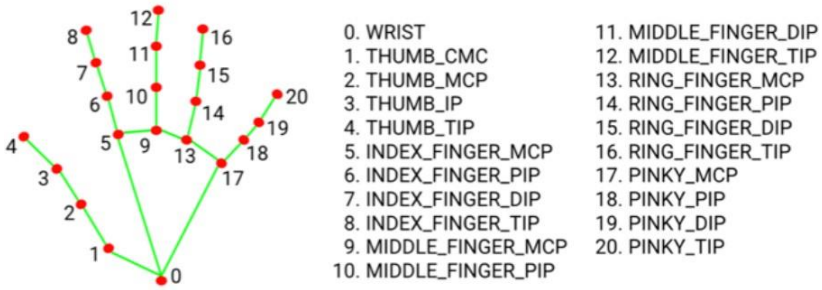

Fig 6.2 Landmarks of a Human Hand

6.2. Stage 2: Data cleaning and normalization

After collecting all the data points under one file, we perform data cleaning and normalization in this stage. Since we only consider the x and y coordinates from the detector, the data file is first checked for any null entries using the pandas library function. Blurry images can sometimes lead to null entries in the dataset, so it is necessary to clean these points to avoid bias in the predictive model.

Any rows containing null entries are searched and removed from the table using their indexes. Once the unwanted points are removed, the x and y coordinates are normalized to fit into our system. Finally, the data file is prepared for splitting into a training and validation set, with 80% of the data reserved for training our model with various optimization and loss functions and 20% for validating the model.

6.3. Stage 3: Prediction using Machine Learning Algorithm

Machine learning techniques are used in this stage to predict the behavior of various hand motions, with the Support Vector Machine (SVM) algorithm being particularly effective. SVM is known to perform well when there is a large gap between classes, making it suitable for categorizing several classes of gestures. Overall, the three stages of this pipeline allow for efficient and accurate hand tracking and gesture recognition using machine learning techniques.

6.4 Algorithm

Step 1: Start
Step 2: Start the webcam video capture and initialize the system.
Step 3: Frame capture with a webcam.
Step 4: Using Media Pipe and OpenCV, detect hands and hand tips and draw hand landmarks and a box around the hand.
Step 5: Draw a rectangle around the computer window area where we'll be using the mouse.
Step 6: Determine which finger is raised.
Step 6.1: The gesture is neutral if all five fingers are up, and the next step is taken.
Step 6.2: The cursor moves to step 2 if both the middle and index fingers are raised.
Step 6.3: A double click is performed when both index and middle fingers are joined side by side, and step2 is performed.
Step 6.4: If both index and middle fingers are down, perform a left click and proceed to step 2.
Step 6.5: If the middle finger is down and the index finger is up, the right click is performed and the process proceeds to step 2.
Step 6.6: Volume up and down are accomplished by joining the thumb and index fingers and moving them up and down.
Step 7: To exit, press the EXIT key

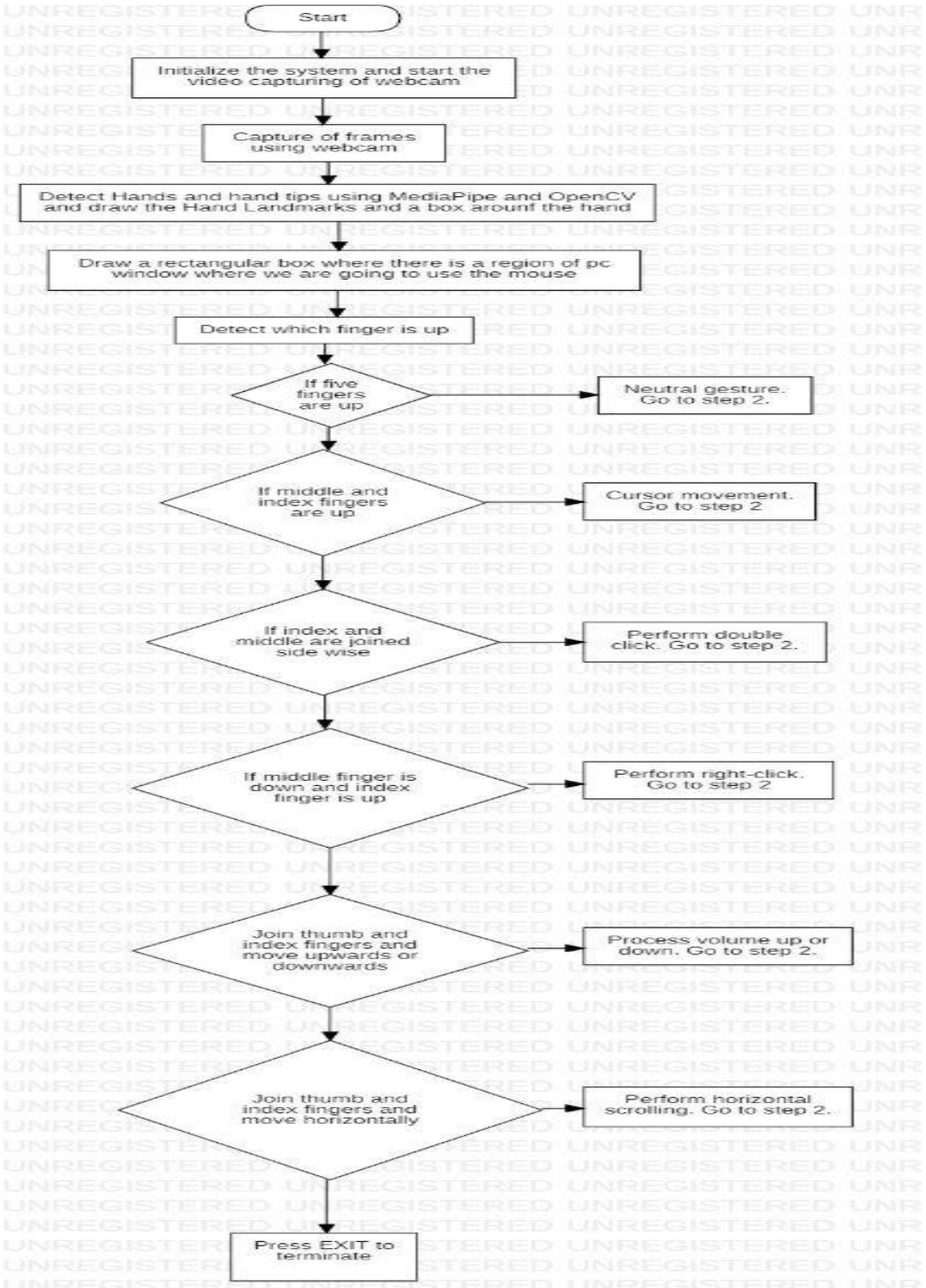

Fig 6.4 General flow of the Algorithm

CHAPTER 7 : EXPERIMENTATION

```
import cv2
import mediapipe as mp
import pyautogui
import math
from enum import IntEnum
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from google.protobuf.json_format import MessageToDict
import screen_brightness_control as sbcontrol
```

Fig 7.1 Importing required modules

7.1 Importing all required modules into our project before we work on anything else. We run the requirements.txt file to automatically install all modules.

The modules include the following:

- CV2: It is an abbreviation for OpenCV, which stands for Open Source Computer Vision Library. It is a popular open-source library used for image and video processing, computer vision, and machine learning applications. cv2 provides a wide range of functions and tools for processing images and videos, including image filtering, edge detection, object recognition, and feature extraction. It also provides support for working with various image and video formats, such as JPEG, PNG, BMP, and MPEG. cv2 is written in C++, but it also provides Python bindings, making it accessible to a wide range of developers and researchers. It is widely used in various industries, including robotics, automotive, healthcare, and security.

- Mediapipe: It is an open-source framework developed by Google that enables developers to build real-time, cross-platform computer vision applications. It provides a comprehensive suite of pre-built and customizable machine learning models, algorithms, and tools that developers can use to create a wide range of computer vision applications, including object detection, facial recognition, hand tracking, and pose estimation. Mediapipe includes a pipeline-based architecture that allows developers to construct complex computer vision workflows using modular building blocks. These building blocks can be combined in a variety of ways to create custom pipelines that can be tailored to specific use cases. Mediapipe also provides pre-built pipelines and machine learning models that developers can use out of the box, which can help reduce development time and improve accuracy.Overall, Mediapipe is a powerful and flexible framework that can help developers create cutting-edge computer vision applications quickly and easily.

- Pyautogui: It is a Python library that provides cross-platform control of the mouse and keyboard, as well as other GUI automation tasks, such as taking screenshots, controlling windows, and manipulating files. With PyAutoGUI, you can automate repetitive tasks, such as filling out forms, clicking buttons, and typing text, by writing Python scripts that interact with your computer's graphical user interface. The library uses various operating system-level APIs to provide platform-specific functionality, and also includes support for keyboard hotkeys, mouse movements and clicks, and pixel-level image recognition. One thing to note about PyAutoGUI is that it requires careful use, as it can potentially cause unintended actions on your system if used incorrectly. Therefore, it is important to thoroughly test your scripts and ensure that they are working as intended before using them on important tasks. Overall, PyAutoGUI is a powerful tool for GUI automation in Python, and can save developers a significant amount of time and effort by automating repetitive tasks.

- Math: It is a built-in Python library that provides a collection of mathematical functions and constants. It provides a way to perform complex mathematical calculations easily in Python. Some of the functions available in the math library include:

i. Trigonometric functions like sine, cosine, and tangent
ii. Logarithmic functions like log, log10, and log2
iii. Exponential and power functions like exp and pow
iv. Constants like pi, e, and tau

- Enum: In Python, the enum module provides support for enumerations, which are a set of symbolic names (members) that represent unique, constant values. Enumerations can be useful in a variety of contexts, such as defining options for a configuration or representing discrete states in a program.

- Ctypes: It is a built-in Python library that provides a foreign function interface (FFI) for calling functions in shared libraries or dynamic-link libraries (DLLs). It allows Python programs to interface with libraries written in other languages like C, C++, and Assembly. Using ctypes, you can load a shared library into memory and call functions defined in the library. The library can be either a system library or a custom library that you've compiled yourself.

- Comtypes: It is a third-party Python library that provides a way to access and interact with Microsoft Windows Component Object Model (COM) components from Python. COM is a Microsoft technology that enables software components to communicate with each other over a network or within a single computer. With comtypes, you can create and use COM objects from Python, call methods on those objects, and access their properties.

- Pycaw: It is a third-party Python library that provides a simple way to control audio on Windows operating systems. It uses the Windows Core Audio API to interact with the audio system and provides a high-level interface to control the volume, mute status, and other properties of audio devices.

- Protobuf: It is a module in the protobuf package that provides support for encoding and decoding Protocol Buffer messages in JSON format. Protocol Buffers are a language-neutral, platform-neutral, extensible way of serializing structured data for use in communications protocols, data storage, and more. They are often used as a faster and more efficient alternative to XML or JSON. With google.protobuf.json, you can easily encode and decode Protocol Buffer messages in JSON format

- screen_brightness_control: It is a third-party Python library that provides a cross-platform way to control the brightness of the screen or display on Windows, macOS, and Linux operating systems. With screen_brightness_control, you can set the brightness level of the screen to a specific value, increase or decrease the brightness level by a specified amount, or get the current brightness level of the screen.

7.2 Calling gestures by their names every time into a line of code is pretty cumbersome. To tackle this we have labeled each one of them to a binary number

```
FIST = 0
    PINKY = 1
    RING = 2
    MID = 4
    LAST3 = 7
    INDEX = 8
    FIRST2 = 12
    LAST4 = 15
    THUMB = 16
    PALM = 31

    # Extra Mappings
    V_GEST = 33
    TWO_FINGER_CLOSED = 34
    PINCH_MAJOR = 35
    PINCH_MINOR = 36

# Multi-handedness Labels
class HLabel(IntEnum):
    MINOR = 0
    MAJOR = 1
```

Fig 7.2 Enum for mapping all hand gesture to binary number

7.3 Here this bock of code executes different commands according to the list of detectable gesture

```
def getpinchylv(hand_result):
        """returns distance beween starting pinch y coord and current hand position y coord."""
        dist = round((Controller.pinchstartycoord - hand_result.landmark[8].y)*10,1)
        return dist

    def getpinchxlv(hand_result):
        """returns distance beween starting pinch x coord and current hand position x coord."""
        dist = round((hand_result.landmark[8].x - Controller.pinchstartxcoord)*10,1)
        return dist

    def changesystembrightness():
        """sets system brightness based on 'Controller.pinchlv'."""
        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():
        """sets system volume based on 'Controller.pinchlv'."""
        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():
        """scrolls on screen vertically."""
        pyautogui.scroll(120 if Controller.pinchlv>0.0 else -120)
```

Fig 7.3 Executes commands according to detected gestures

7.4 Set of attributes which are defined under the constructor.

```
tx_old : int
        previous mouse location x coordinate
    ty_old : int
        previous mouse location y coordinate
    flag : bool
        true if V gesture is detected
    grabflag : bool
        true if FIST gesture is detected
    pinchmajorflag : bool
        true if PINCH gesture is detected through MAJOR hand,
        on x-axis 'Controller.changesystembrightness',
        on y-axis 'Controller.changesystemvolume'.
    pinchminorflag : bool
        true if PINCH gesture is detected through MINOR hand,
        on x-axis 'Controller.scrollHorizontal',
        on y-axis 'Controller.scrollVertical'.
    pinchstartxcoord : int
        x coordinate of hand landmark when pinch gesture is started.
    pinchstartycoord : int
        y coordinate of hand landmark when pinch gesture is started.
    pinchdirectionflag : bool
        true if pinch gesture movment is along x-axis,
        otherwise false
    prevpinchlv : int
        stores quantized magnitued of prev pinch gesture displacment, from
        starting position
    pinchlv : int
        stores quantized magnitued of pinch gesture displacment, from
        starting position
    framecount : int
        stores no. of frames since 'pinchlv' is updated.
    prev_hand : tuple
        stores (x, y) coordinates of hand in previous frame.
    pinch_threshold : float
        step size for quantization of 'pinchlv'.
```

Fig 7.4 Set of attributes in the above block of code

7.5 Obtain landmarks from mediapipe, entry point for whole program.

```
class GestureController:

    gc_mode = 0
    cap = None
    CAM_HEIGHT = None
    CAM_WIDTH = None
    hr_major = None # Right Hand by default
    hr_minor = None # Left hand by default
    dom_hand = True

    def __init__(self):
        """Initilaizes attributes."""
        GestureController.gc_mode = 1
        GestureController.cap = cv2.VideoCapture(0)
        GestureController.CAM_HEIGHT = GestureController.cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
        GestureController.CAM_WIDTH = GestureController.cap.get(cv2.CAP_PROP_FRAME_WIDTH)

    def classify_hands(results):
        """
        sets 'hr_major', 'hr_minor' based on classification(left, right) of
        hand obtained from mediapipe, uses 'dom_hand' to decide major and
        minor hand.
        """
        left , right = None,None
        try:
            handedness_dict = MessageToDict(results.multi_handedness[0])
            if handedness_dict['classification'][0]['label'] == 'Right':
                right = results.multi_hand_landmarks[0]
            else :
                left = results.multi_hand_landmarks[0]
        except:
            pass
        try:
            handedness_dict = MessageToDict(results.multi_handedness[1])
            if handedness_dict['classification'][0]['label'] == 'Right':
                right = results.multi_hand_landmarks[1]
            else :
                left = results.multi_hand_landmarks[1]
        except:
            pass

        if GestureController.dom_hand == True:
            GestureController.hr_major = right
            GestureController.hr_minor = left
        else :
            GestureController.hr_major = left
            GestureController.hr_minor = right
```

Fig 7.5 Handles camera, obtain landmarks from mediapipe, entry point for whole program.

7.6 Entry point of whole program, captures video frame and passes, obtains landmark from mediapipe and passes it to 'handmajor' and 'handminor'

```
def start(self):

        handmajor = HandRecog(HLabel.MAJOR)
        handminor = HandRecog(HLabel.MINOR)

        with mp_hands.Hands(max_num_hands = 2,min_detection_confidence=0.5, min_tracking_confidence=0.5) as hands:
            while GestureController.cap.isOpened() and GestureController.gc_mode:
                success, image = GestureController.cap.read()
                if not success:
                    print("Ignoring empty camera frame.")
                    continue

                image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
                image.flags.writeable = False
                results = hands.process(image)

                image.flags.writeable = True
                image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
                if results.multi_hand_landmarks:
                    GestureController.classify_hands(results)
                    handmajor.update_hand_result(GestureController.hr_major)
                    handminor.update_hand_result(GestureController.hr_minor)
                    handmajor.set_finger_state()
                    handminor.set_finger_state()
                    gest_name = handminor.get_gesture()
                    if gest_name == Gest.PINCH_MINOR:
                        Controller.handle_controls(gest_name, handminor.hand_result)
                    else:
                        gest_name = handmajor.get_gesture()
                        Controller.handle_controls(gest_name, handmajor.hand_result)

                    for hand_landmarks in results.multi_hand_landmarks:
                        mp_drawing.draw_landmarks(image, hand_landmarks, mp_hands.HAND_CONNECTIONS)
                else:
                    Controller.prev_hand = None
                cv2.imshow('Gesture Controller', image)
                if cv2.waitKey(5) & 0xFF == 13:
                    break
        GestureController.cap.release()
        cv2.destroyAllWindows()
gc1 = GestureController()
gc1.start()
```

Fig 7.6 Entry point of whole program

CHAPTER 8 : TESTING AND RESULTS

8.1 Test cases:

8.1.1 To perform No Action on the Screen, as shown in Fig-4.1, if all of the fingers are up with tip Id = 0,1,2, 3, and 4, the computer is set to not perform any mouse events on the screen.

Fig 8.1.1 Neutral Gesture

8.1.2 For navigating the computer window with the mouse cursor. The mouse cursor is made to move around the computer window using Python's AutoPy package if the index finger with tip Id = 1 and the middle finger with tip Id = 2 are up, as shown in Fig-8.1.2

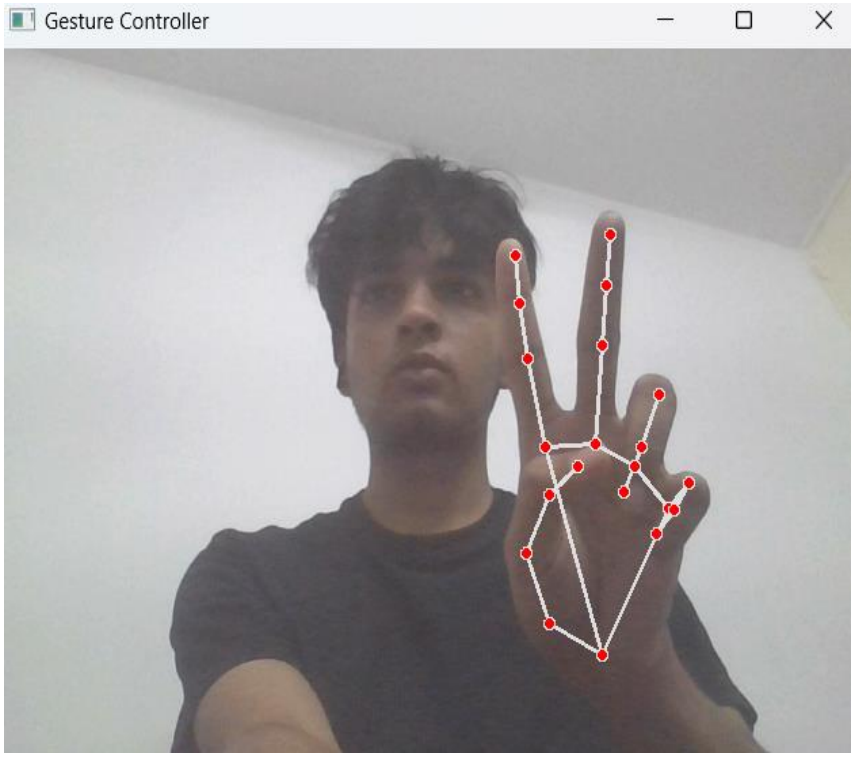

Fig 8.1.2 Cursor Control

8.1.3 To perform a left-button click with the mouse. The computer is made to perform the left mouse button click using the pynput Python package if both the index finger with tip Id = 1 and the middle finger with tip Id = 2 are up and the distance between the two fingers is less than 30px, as shown in Fig-8.1.3

Fig 8.1.3 Left Click

8.1.4 To use the mouse's right-button click. The computer is programmed to perform the right mouse button click if both the index finger with tip Id = 1 and the middle finger with tip Id = 2 are up and the distance between the two fingers is less than 40 px, as shown in Fig-8.1.4

Fig 8.1.4 Right Click

8.1.5 If both the index and middle fingers are closed, the mouse will perform a drag operation. As shown in Fig-4.5, the computer is programmed to perform the drag operation button click using the pynput Python package.

Fig 8.1.5 Drag Operation

8.2 Discussion of the above test cases:

Finally the above test cases describe a system that uses hand gestures to interact with a computer in a more natural and intuitive way than traditional mouse and keyboard inputs. The system is capable of performing various mouse operations based on the position and movement of the fingers, allowing the user to control the computer without physically touching a mouse or trackpad.

- To perform No Action on the Screen, all fingers must be up with tip Ids of 0, 1, 2, 3, and 4. This essentially tells the computer not to perform any mouse events on the screen.

- To navigate the computer window with the mouse cursor, the index finger with tip Id = 1 and the middle finger with tip Id = 2 must be up. The system uses Python's AutoPy package to move the mouse cursor around the computer window.

- To perform a left-button click with the mouse, the index finger with tip Id = 1 and the middle finger with tip Id = 2 must be up and the distance between the two fingers must be less than 30px. The system uses the pynput Python package to perform the left mouse button click.

- Similarly, to use the mouse's right-button click, the index finger with tip Id = 1 and the middle finger with tip Id = 2 must be up and the distance between the two fingers must be less than 40px. The system is programmed to perform the right mouse button click in this scenario.

- Finally, if both the index and middle fingers are closed, the system performs a drag operation using the pynput Python package. This allows the user to click and hold the mouse button while dragging an object or selecting text, for example.

- Overall, the system represents a novel way of interacting with a computer and has the potential to enhance productivity and efficiency by reducing the need for traditional mouse input.

CONCLUSION

In conclusion our proposed gesture-controlled virtual mouse system using the Mediapipe framework offers a range of features that make it a promising new way to interact with computers. One of the key advantages of our system is its high accuracy in recognizing hand gestures, thanks to the advanced computer vision algorithms provided by Mediapipe. This allows for precise control over mouse movements and operations, which is essential for many tasks such as graphic design or gaming.

Additionally, our system is designed to work in low light conditions, that can detect hand movements even in dimly lit environments. This makes it ideal for use in a range of settings, from home offices to gaming environments.
Our system also includes a range of features for performing different mouse operations, such as left-click, right-click, and drag. This allows for a wide range of interactions with the computer, and eliminates the need for traditional mouse and keyboard inputs.

Overall, we believe that our proposed gesture-controlled virtual mouse system using the Mediapipe framework has the potential to revolutionize the way that people interact with computers. By offering a more precise, natural way to control mouse movements and operations, our system could help to improve productivity, reduce physical strain on users, and make computing more accessible to a wider range of people. With continued development and refinement, we believe that this technology will become increasingly prevalent in the years to come, and will play an important role in shaping the future of human-computer interaction.

REFERENCES

[1] Horatiu -Stefan Grif, Cornel Cristian Farcas. 2016. Mouse cursor controlled system based on hand gestures. Procedia Technology, Volume 22.

[2] Horatiu -Stefan Grif, Trian Turc. 2018. Cursor Control using Hand Gesture Recognition. Procedia Manufacturing, Volume 22.

[3] Ketan Gomase, Akshata Dhanawade. 2022. Sign language Recognition using Mediapipe. International Research Journal of Engineering and Technology.

[4] Murakami K, Taguchi H. 1991. Gesture recognition using recurrent neural networks. In: Proceedings of the ACM SIGCHI conference on Human factors in computing systems, pp 237–242. https ://dl.acm.org/doi/pdf/10.1145/10884 4.10890 0

[5] Wang RY, Popović J. 2009. Real-time hand-tracking with a color glove. ACM Trans Graph 28(3):63

[6] Rekha J, Bhattacharya J, Majumder S. 2011. Hand gesture recognition for sign language: a new hybrid approach. In: International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV), pp 80–86

[7] Kurdyumov R, Ho P, Ng J. 2011. Sign language classification using webcam images, pp 1–4. http://cs229 .stanf ord.edu/proj2 011/ Kurdy umovHoNg-SignLanguageClassificatio nUsin gWebc amIma ges.pdf

[8] Tharwat A, Gaber T, Hassanien AE, Shahin MK, Refaat B. 2015. Sift-based arabic signlanguage recognition system. In: Springer Afro-European conference for industrial advancement, pp359–370. https ://doi.org/10.1007/978-3-319-13572-4_30

[9] Baranwal N, Nandi GC. 2017. An efficient gesture based humanoid learning using wavelet descriptor and MFCC techniques. Int J Mach Learn Cybern 8(4):1369–1388

[10] Elakkiya R, Selvamani K, Velumadhava Rao R, Kannan A. 2012. Fuzzy hand gesture recognition based human computer interface intelligent system. UACEE Int J Adv Comput Netw Secur 2(1):29–33 (ISSN 2250–3757)

[11] Ahmed AA, Aly S. 2014. Appearance-based arabic sign language recognition using hidden markov models. In: IEEE International Conference on Engineering and Technology (ICET), pp 1–6. https ://doi.org/10.1109/ICEng Techn ol.2014.70168 04

[12] R. Sharma, R. Khapra, N. Dahiya. June 2020. Sign Language Gesture Recognition, pp.14-19

[13] W. Liu, Y. Fan, Z. Li, Z. Zhang. Jan 2015 . Rgbd video based human hand trajectory tracking and gesture recognition system in Mathematical Problems in Engineering,

[14] Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C. L., & Grundmann, M. 2020. MediaPipe Hands: On-device Real-time Hand Tracking. arXiv preprint arXiv:2006.10214.

[15] Das, P., Ahmed, T., & Ali, M. F. 2020, June. Static Hand Gesture Recognition for American Sign Language using Deep Convolutional Neural Network. In 2020 IEEE Region 10 Symposium (TENSYMP) (pp. 1762-1765). IEEE.

[16] M. Taskiran, M. Killioglu and N. Kahraman. 2018 . A Real-Time System for Recognition of American Sign Language by using Deep Learning, 2018 41st International Conference on Telecommunications and Signal Processing (TSP), Athens, Greece, pp. 1-5, doi: 10.1109/TSP.2018.8441304.

[17] Nazmus Saquib and Ashikur Rahman. 2020. Application of machine learning techniques for real-time sign language detection using wearable sensors. In Proceedings of the 11th ACM Multimedia Systems Conference (MMSys '20). Association for Computing Machinery, New York, NY, USA, 178–189. DOI:https://doi.org/10.1145/3339825.3391869

[18] Dutta, K. K., & Bellary, S. A. S. 2017, September. Machine learning techniques for Indian sign language recognition. In 2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC) (pp. 333-336). IEEE.

[19] Sahoo, Ashok. 2014. Indian sign language recognition using neural networks and kNN classifiers. Journal of Engineering and Applied Sciences. 9. 1255-1259.

[20] Raheja JL, Mishra A, Chaudary A. 2016 September . Indian Sign Language Recognition Using SVM 1. Pattern Recognition and Image Analysis.; 26(2).

[21] Pigou, L., Dieleman, S., Kindermans, P. J., & Schrauwen, B. 2014, September. Sign language recognition using convolutional neural networks. In European Conference on Computer Vision (pp. 572-578). Springer, Cham.