Project Report

On

” Translating hand gesture into control action.”

In fulfillment of requirements for the award of the degree in
Master of Computer Applications 2023

Master of Computer Applications

Submitted by

Abid Raza
Roll No: 21MCA001.

Nima Lamu Tamang
Roll No. 21MCA010.

Shuvam Roy
Roll No. 21MCA016.

Under the guidance of

Dr. Mohan Pratap Pradhan
Associate Professor
Head Department of Computer Applications
Sikkim University, Gangtok-737102, India

Department of Computer Applications
Sikkim University
2023

Contents

Acknowledgements

It was our privilege and honor to work under the supervision of Dr. Mohan Pratap Pradhan, Associate Professor, Head of Department, Computer Applications, Sikkim University, Gangtok, Sikkim.

We would like to express our sincere gratitude to our supervisor Dr. Mohan Pratap Pradhan for his excellent guidance, care, patience, and for providing us with an excellent atmosphere for doing research.

We would like to thank all faculty members of the Department of Computer Applications, Sikkim University, for their valuable suggestions during our research work.

We would like to thank the staffs of the Department of Computer Applications, Sikkim University, whowere always willing to help and give their best support in our project work journey.

We deeply acknowledge the love, cooperation, and the moral support extended by our parents, friends, relatives, and colleagues right from the beginning of project work. Finally, We must thank the Almighty for blessing us in this journey.

Declaration

I **Abid Raza**, Roll No.**21MCA001**, a registered candidate Of Master of Computer Applications(MCA) under Department of Computer Applications, Sikkim University, declare that this is my own original work and does not contain material for which the copyright belongs to a third party and that it has not been presented and will not be presented to any other University/Institute for a similar or any other Degree award.

I further confirm that for all third-party copyright material in my project report (including any electronic attachments), I have" blanked out" third party material from the copies of the thesis fully referenced the deleted materials and where possible, provided links (URL) to electronic sources of the material.

I hereby transfer exclusive copyright for this project report to Sikkim University. The following rights are resewed by the author:

a) The right to use, free of charge, all or part of this article in future work of their own, such as books and lectures, giving reference to the original place of publication and copyright holding.

b) The right to reproduce the article or thesis for their own purpose provided the copies are not offered for sale.

Abid Raza
Roll No: 21MCA001
Date: 12/06/2023

Declaration

I **Nima Lamu Tamang**, Roll No.**21MCA010**, a registered candidate Of Master of Computer Applications(MCA) under Department of Computer Applications, Sikkim University, declare that this is my own original work and does not contain material for which the copyright belongs to a third party and that it has not been presented and will not be presented to any other University/Institute for a similar or any other Degree award.

I further confirm that for all third-party copyright material in my project report (including any electronic attachments), I have" blanked out" third party material from the copies of the thesis fully referenced the deleted materials and where possible, provided links (URL) to electronic sources of the material.

I hereby transfer exclusive copyright for this project report to Sikkim University. The following rights are resewed by the author:

c) The right to use, free of charge, all or part of this article in future work of their own, such as books and lectures, giving reference to the original place of publication and copyright holding.

d) The right to reproduce the article or thesis for their own purpose provided the copies are not offered for sale.

Nima Lamu Tamang
Roll No: 21MCA010
Date: 12/06/2023

Declaration

I **Shuvam Roy**, Roll No.**21MCA016**, a registered candidate Of Master of Computer Applications(MCA) under Department of Computer Applications, Sikkim University, declare that this is my own original work and does not contain material for which the copyright belongs to a third party and that it has not been presented and will not be presented to any other University/Institute for a similar or any other Degree award.

I further confirm that for all third-party copyright material in my project report (including any electronic attachments), I have" blanked out" third party material from the copies of the thesis fully referenced the deleted materials and where possible, provided links (URL) to electronic sources of the material.

I hereby transfer exclusive copyright for this project report to Sikkim University. The following rights are resewed by the author:

e) The right to use, free of charge, all or part of this article in future work of their own, such as books and lectures, giving reference to the original place of publication and copyright holding.

f) The right to reproduce the article or thesis for their own purpose provided the copies are not offered for sale.

Shuvam Roy
Roll No: 21MCA016
Date: 12/06/2023

Certificate from the Supervisor

This is to certify that Mr. **Abid Raza**, Roll No. **21MCA001, Nima Lamu Tamang** Roll No. **21MCA010** and **Shuvam Roy**, Roll No. **21MCA016** are registered candidate Of Master of Computer Applications Programme under the **Department of Computer Applications** of Sikkim University.

The undersigned certifies that they have completed all other requirements for submission of the dissertation and hereby recommends for the acceptance of their project entitled **'translating hand gesture into control action'** in the partial fulfillment of the requirements for the award of MCA Degree by Sikkim University, Gangtok, India.

Dr. Mohan Pratap Pradhan
Associate Professor
Head of Department Computer Applications
Sikkim University
Gangtok-737102,
IndiaDate:
12/06/2023

Abstract

The way humans communicate has also takena significant leap.

Hand gesture recognition is of great importance for Human Computer interaction (HCI) because of its extensive applications in virtual reality and sign language recognition etc. Since hand gestures are able to express enriched information, the hand gesture recognition is widely used in robot control, intelligent furniture and other aspects. Hand gesture recognition is one of the most researched topics and has made significant progress in terms of accuracy. Using the bare minimum hardware available today, we come to show how hand gesturecan be used to generate more than a thousand different commands which will bevery useful for further research and development.

List of Tables

Chapter 1

Introduction

The establishment of reliable communication services and access to the World Wide Web has also added to the value and quality of the services. Human-Machine Interaction (HMI) is one such initiative that for some time has been gaining tremendous ground because of its ease of use and assistance it provides to the wide categoriesof end-users.

1.1 Motivation

Devices used in the health sector can use this technology to reduce contact, especially the devices used in critical areas where chances of infection may increase while using traditional ways of interaction.

Contactless control has been highly researched in the present age. With the rise in highly communicable diseases like we have seen in recent years and the increase in population density, it becomes a necessity for us Humans to decrease our contact traces in public/open devices.

1.2 Proposed System

In this work, we aim to develop a contact-free interface technology that is a computationally simple, portable, and economically viable solution for enabling control actions using a computer system with the aid of image processing techniques and pattern recognition. The goal will be to make use of the hardware present in the devices used by the masses such as laptops which in today's market come with a built-in webcam. Also, bring into use the available software technology for the detection and calculation of human hand data.

1.3 Objective

.In this work we are aiming to develop a contact-free interface technology between a human and computer.

.To understand the current state of affairs in hand gesture recognition andlearn about their limitations and then suitably propose improvised solutions.

.Design and develop a solution to translate hand gestures into control actionsusing a webcam

.Associate a set of well-defined control actions with the detected hand gestures.

.

Chapter 2

Literature Review

2.1 Introduction

Gesture recognition is a technology that uses sensors to read and interpret hand movements as commands

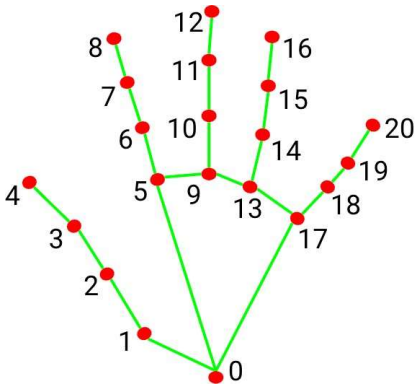
Hand gesture recognition is the process of detecting a hand in a frame either from web camera or from the image file. Hand gesture recognition system has manifold of applications and the ability to interact with machine efficiently this created a massive amount of attention towards hand recognition system

There has been a lot of research in the field of Hand gesture Recognition, re- searchers can now be seen using methods such as deep learning, which opens upa lot of different types of architecture, as compared to traditional hand detection methods which use artificially designed weak features, such as skin color and shape features. CNN-based detection methods are becoming a popular research topic in computer vision field recently, because higher-level deep features can be learned from the networks. There are many solutions and frameworks available for Hand recognition which has been able to harness the power of CPU as well as GPU for processing the image, in mobile devices as well.

2.1.1 MediaPipe Hands

Media pipe is used to detect a hand landmark.MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame. MediaPipe Hands utilizes an ML pipeline consisting of multiple models working together:
1) A palm detection model that operates on the full image and returns an oriented hand bounding box.

2) A hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand keypoints.
Providing the accurately cropped hand image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy. With the help of MediaPipe Hands, we are ableto get the landmark values of the hands, with these values we are able to detect the significant changes in the hand gesture.



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

Figure 2.1: Hand Landmarks

Chapter 3

Methodology

3.1 Image Processing

To read hand data we do require some software component to have access to the webcam and give us the real-time image data. For this task of image processing, we have used OpenCV (Open Source Computer Vision Library). With the help of OpenCV, we are able to get the real-time Image data from the webcam.

3.2 Computer Vision

For recognition of the hand and making use of the data we have incorporated mediapipe. Mediapipe is an open-source cross-platform, customizable ML solution for live and streaming media. Mediapipe offers many solutions of which we have incorporated MediaPipe Hands.

3.3 Control actions

With the detection of changes in key hand points with respect to other points would trigger specific controls/commands. These commands can be used to control the device and give specific commands. There can be total of 1024 different hand gesture combination we can retrieve in this project.

3.4 Software requirement

For the development of this project the following are the software and tools used to get the result that we wanted.

IDE	VSCODE 1.79
Language	Python 3.7
Python libraries	OpenCV,cvzone, Mediapipe, autopy, pyautogui, NumPy.

Table 3.1: Software Requirement

3.5 Hardware requirement

Here we have the minimum hardware requirement for running the project that we have developed.

Webcam
Memory 4GB RAM or above
Hard Disk 500 GB or above
Processor i3 core processor or above

Table 3.2: Hardware Requirement

3.6 Time plan

The work is proposed to be undergone in the following phases:

- Phase I: Planning
- Phase II: Literature survey
- Phase III: Gathering requirements and analysis.
- Phase IV: Coding and integration
- Phase V: Testing.
- Phase VI: Documentation.

Project Timeline		
Stage	**Phase**	**Duration**
I	Planning	1st April - 10th April
II	Literature survey	11th April - 25th April
III	Gathering requirements and analysis	26th April - 5th May
IV	Coding and Integration	6th May - 26th May
V	Testing	27th May - 6th June
VI	Documentation	7th June - 25th June

Table 3.3: Project Timeline

Chapter 4

Implementation

4.1 Process

To achieve our goal, we have to go through the following steps:

- To get the image data from the peripheral device.
- To process the image and detect the hands.
- Use the landmark of detected hand to capture real-time changes in hand movements.
- Use those to call in controlled actions.

4.2 Identification of Hands

With the use of Mediapipe and OpenCV we are able to locate the hand and its 21 landmark points. These landmark points have their own coordinate values representing their location in the image space. On the image space the top left corner is takes as the origin, everything to the left is represented with the x-axis and everything downward is represented with the y-axis. These landmark points are vital as they provide the base data on the basis of which identification of hand and command detection work.

Mediapipe provided the coordinates for both the hands detected in the sample space but did not provide a concrete data to identify the hand. This caused some drawbacks in our goal of detecting hand gestures as we were not ableto differentiate the hand and were provided with repeated values.

4.2.1 Left hand detection

For the detection of the left hand we identified its specific characteristics in the image space. We identified that the pinkey finger represented by the landmark 20, its x-coordinate value is always less than the x-coordinate of the tip of the thumb represented by landmark 4. This became one of the criteria to identify a left hand.

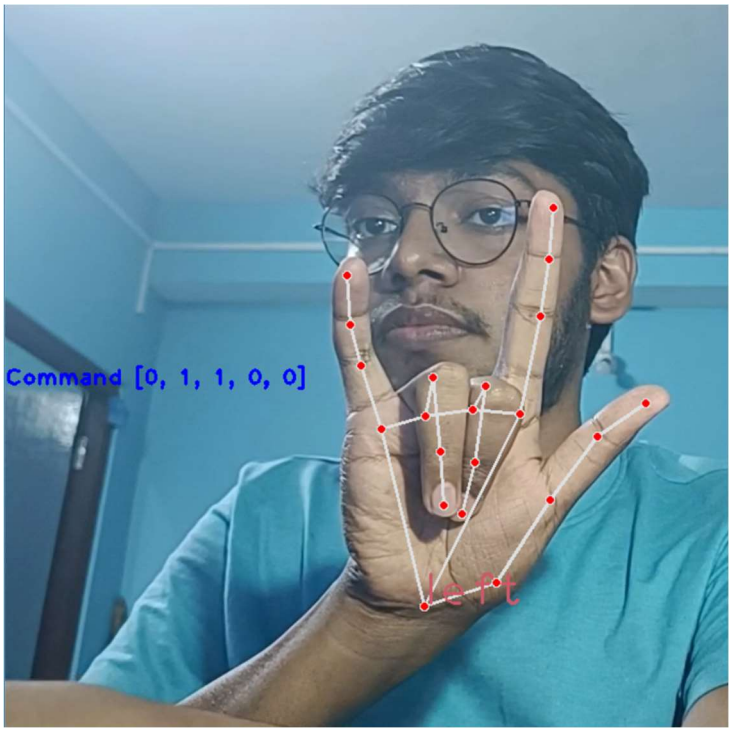


Figure 4.1: Left hand command

4.2.2 Right hand detection

Similarly For the detection of the right hand we identified its specific characteristics in the image space. We identified that the pinky finger represented by landmark 20, its x-coordinate value is always greater than the x-coordinate of the tip of the thumb represented by landmark 4. This became one of the criteria to identify a right hand.

Figure 4.2: Right hand command

4.2.3 Orientation Detection

To check the orientation of the hand if the y-coordinate of the tip of the middle- finger represented by landmark 12 is less than the y-coordinate of the base of the palm represented by landmark 0 in fig. 2.1, then hand fulfils the criteria of beingin upright orientation.

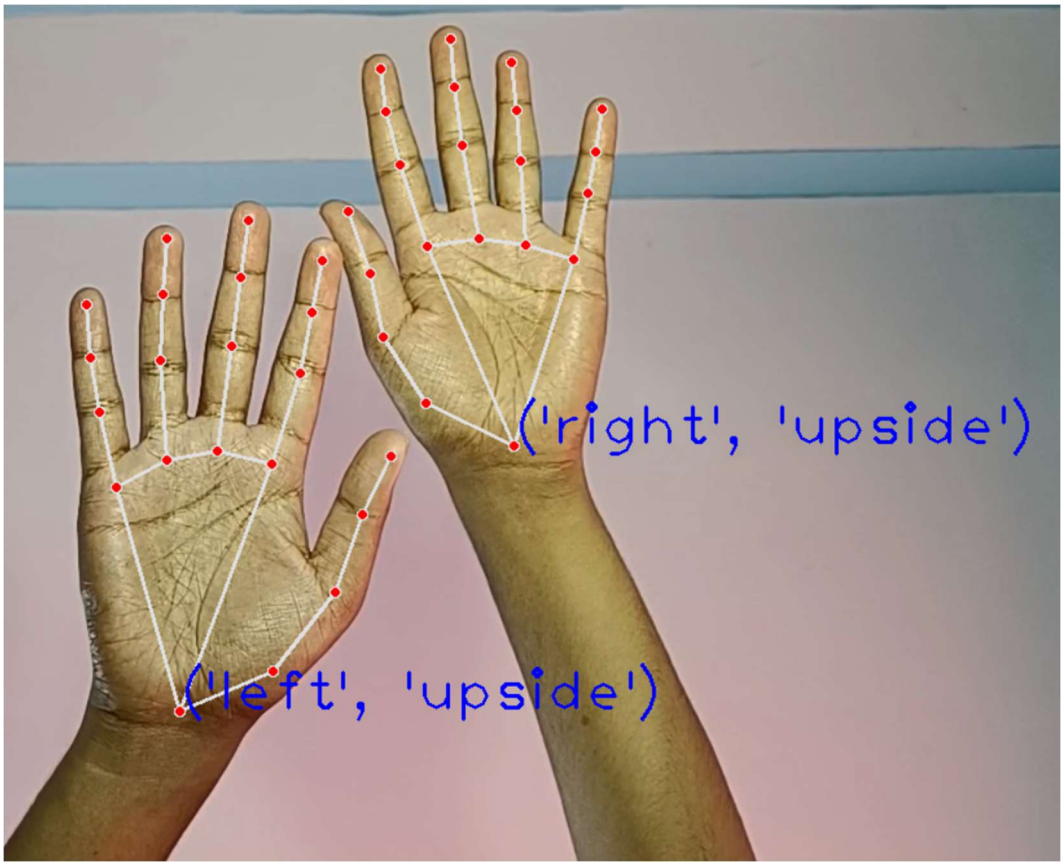


Figure 4.3: Upright Orientation

Similarly, if the y-coordinate of the tip of the middle-finger represented by land- mark 12 is greater than the y-coordinate of the base of the palm represented by landmark 0 in fig. 2.1, then hand fulfils the criteria of being in upside down ori- entation.

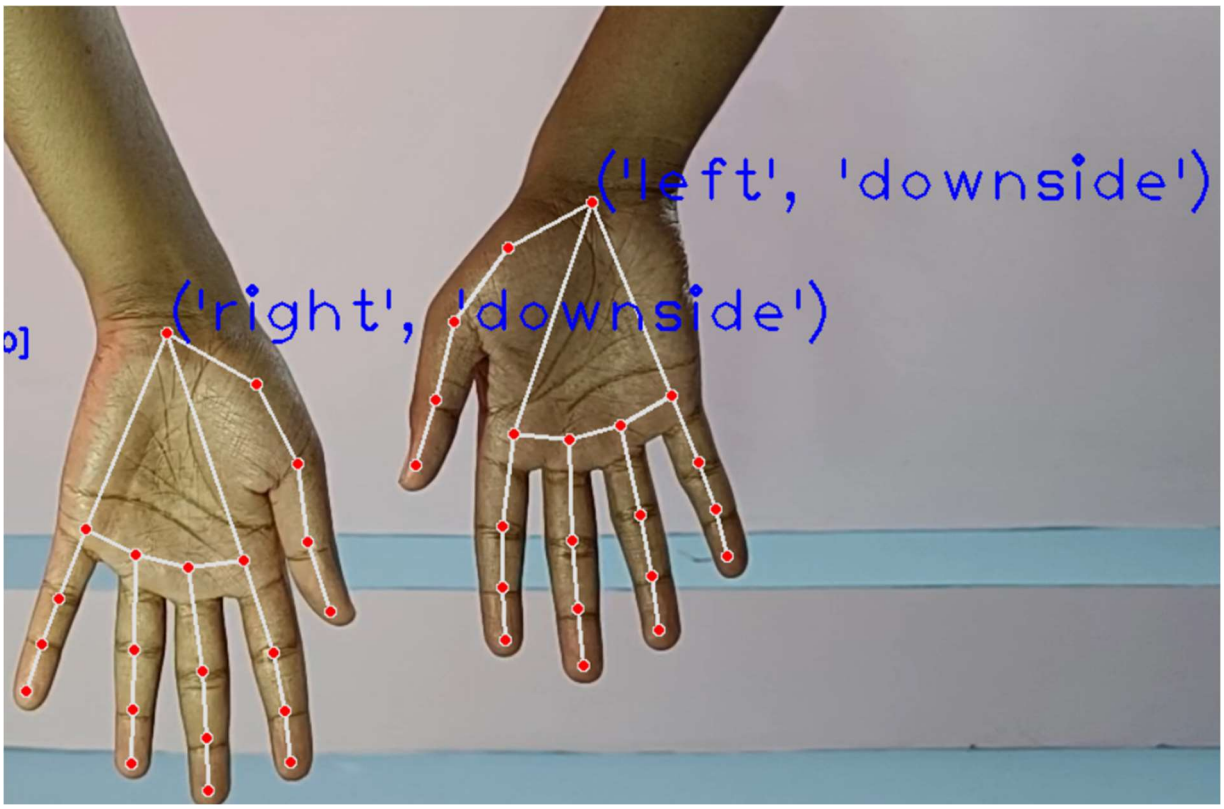


Figure 4.4: Upside-down Orientation

4.3 Hand gestures detection

Here we have take up open finger and closed finger as the gesture to build up the commands that would be interpreted into actions. With each finger having theability to give out two commands, with the combination of 10 different finger we have a vast array of commands at disposal to be assigned a task.

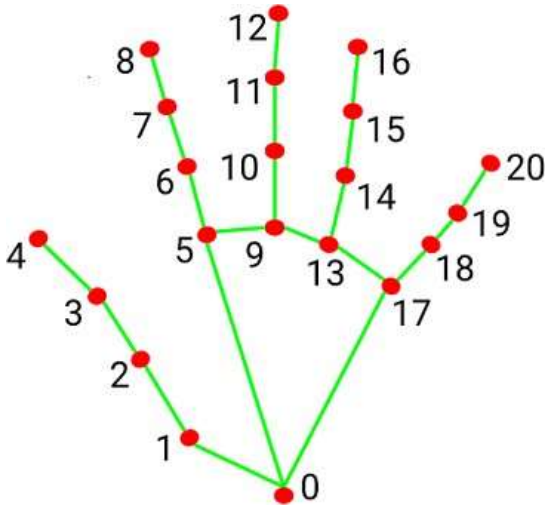


Figure 4.5: Hand Landmarks

Finger Interpretation

For the detection of closed finger, we have taken up the landmark of the tip of each finger and set a particular criteria in which if the y-coordinate of the tip landmark becomes less than a particular landmark below the tip then a closed finger can be said to be detected.

Figure 4.6: Closed finger

Thumb Interpretation

Similar to the finger, for the detection of closed thumb we have take up the landmark of the tip of thumb set a particular criteria.

```
def thumb(fing,coordinate,side,orient):
 axis =0
 if orient == "upside":
    upper = 0
    lower = 1
 else:
    upper = 1
    lower = 0
 tip = coordinate[fing[upper]][axis]
 base = coordinate[fing[lower]][axis]
 if side == "left":
    if tip > base:
       return 0
    return 1
 else:
    if tip < base:
       return 0
    return 1
```

Figure 4.7: Thumb detection

in which if the x-coordinate of the tip of the thumb becomes less or more (depending on the hand) than a particular landmark towards the palm then a close thumb can be said to be detected.

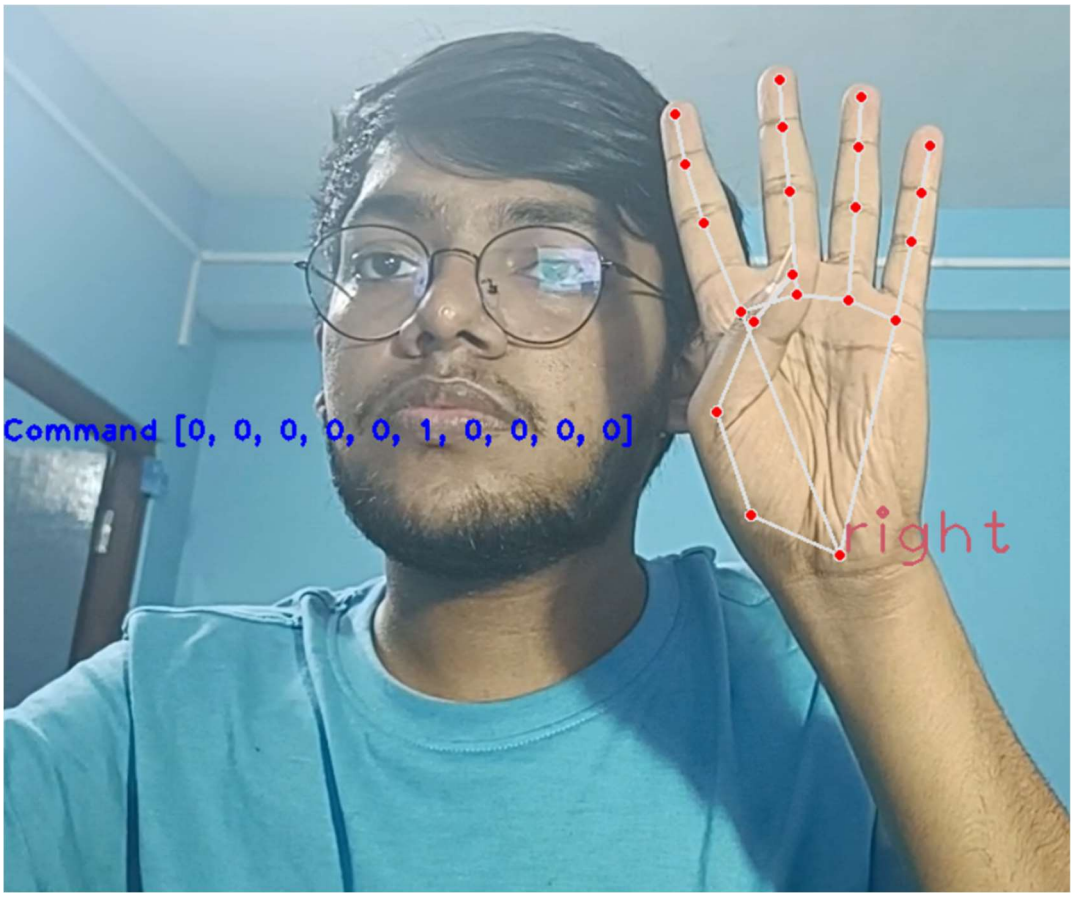


Figure 4.8: Thumb detection

Chapter 5

Command and Trigger

With 10 different fingers giving two signals, we can get a vast array of commands. In this project, we have taken the open finger to give out the signal of 0 and the closed finger the signal of 1. This gives us the ability to combine and create custom commands. By combining both hands we were able to create a list,showing the real-time combination.

5.1 Detection of command

With the help of mediapipe, we were able to get the real-time landmark coordinatesper picture frame. From each finger, we had taken two landmarks to detect the changes, so in total, we had a pair of 5 combinations. To interpret the data we loop through the 5 combinations and see the appropriate difference in the y-coordinate for the finger and the x-coordinate for the thumb. If the finger or thumb is found to be closed we map the appropriate fingers index in the command array with 1or else with 0.

Media pipe does give us the ability to find out the number of hands in the frame, so if there are two different hands we get two different coordinate datasets. For two hands the loop is run two times. If only a single hand is detected in the frame then for the missing hand the default value is given ie. an array of all zeroes. Ifno hands are detected then the command array is set to the default value whichis an array of all zeros.

5.2 Triggering task

For the task to be executed the command combination has to match the one mapped out with the appropriate task in the command task table. Since we are tracking real-time data from the webcam cam a single command could trigger aseries of the same task if mapped out directly, hence the need for a trigger was a must.

```
if combicommand != command_call[1]:
    command_call[1] = combicommand

    if command_call[0] != command_call[1]:
       print("both", command_call[0], command_call[1])

       if check_availableCommand(command_call[0]):
          fire_function(command_call[0])
       else:
          print("On this sign, we don't have an action")

       command_call[0] = command_call[1]
       command_call[1] = command_call[0]
```

Figure 5.1: Trigger block

5.2.1 Command queue

The command queue is a list that is set to hold a max of two lists of commands.The commands once detected are filled in the command queue. For the commands to be inside the command queue the commands have to be unique and not thesame as the last command entered. This stops the redundancy of commands inthe command queue.

```
if combicommand != command_call[1]:
      command_call[1] = combicommand
```

Figure 5.2: Command Queue

5.2.2 Working of trigger

We have a set of default values set to an array of all zeros that can be given manually by the user or is set if no hands are detected in the frame.

```
def fire_function(val):
 comm = get_key(val, command_function_identification)
 comm()
```

Figure 5.3: Function calling task

A trigger is executed if the commands in the command queue is not equal and the first command is followed by the default command. The trigger checks the command in the command queue and if the condition is fulfilled then the program enters the trigger. The trigger only checks if the criteria has been fulfilled by the commands in the command queue.

5.2.3 Execution of task

Once the criteria of the trigger is fulfilled the trigger queue values go through some more functions. Out of the two commands the one command other than the default is checked if it is available in the list of commands that has been mapped out. If it's true, then the appropriate function is executed.

Chapter 6

Virtual Keyboard

6.1 Introduction:

A virtual keyboard allows users to type or input text using gestures, movements, or other forms of input without the need for a physical keyboard. The virtual keyboard typically relies on computer vision technologies, such as image or video processing, to track and analyze user movements or gestures.
It captures input data from the user, such as hand movements or gestures, and interprets them to identify the intended characters or commands.
We have defined a specific command for our virtual keyboard by which we can get the virtual keyboard using hand gestures only. When the given hand gesture matches with the command array then the specified command gets executed.

6.2 Enable Virtual Keyboard by specific hand-gestures command:

```
def command4():
    print("Running virtual keyboard")
   # run_virtual_keyboard(img,coordinatelist)
    keyprint()
```

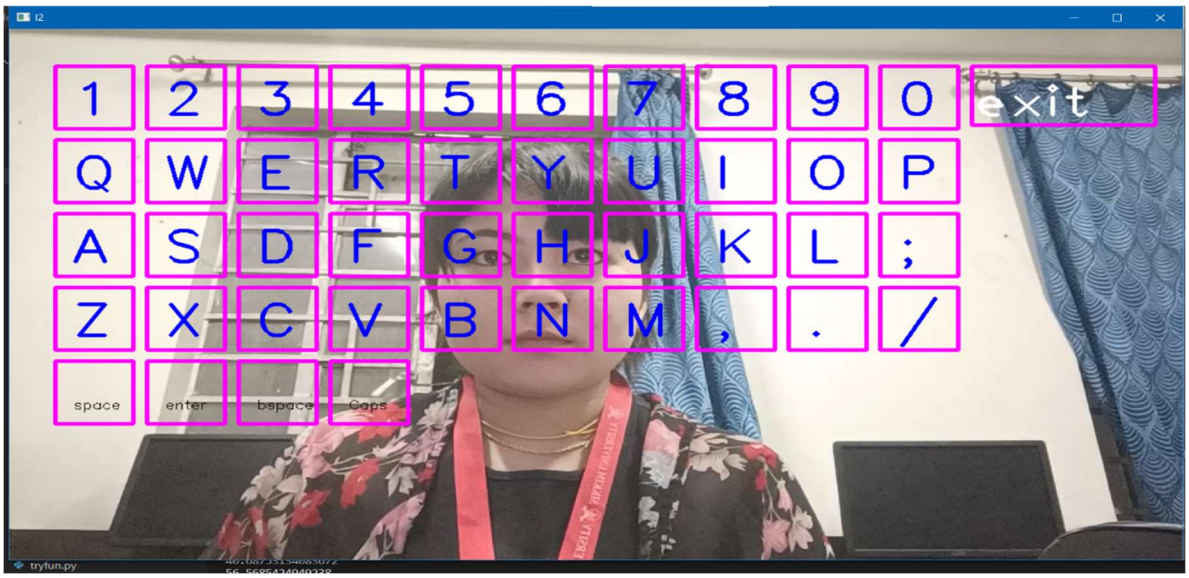
```
# command dictionary
command_function_identification = {
    command1: [1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
    command2: [1, 1, 0, 0, 1, 0, 0, 0, 0, 0],
    command3: [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    command4: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
}
```

6.3 Printing Keys

We create a two dimensional array for all keys that we are going to show on our camera view and we determine for the position of each key , position of all key store in a button list and we enumerate position of all key with key in button list.

So that each key associate with position and by using this our drawAll function. We print all keys on our camera using opecnCv- putText function.

```
def drawAll(img, buttonList):
    for i, button in enumerate(buttonList):
        x, y = button.pos
        w, h = button.size
        if button.text in ["space", "enter", "bspace"]:
            font_scale = .56  # Decrease font size for new buttons
            boldness = 1
            color =(0, 0, 0)
        else:
            font_scale = 2.0  # Default font size for regular buttons
            boldness = 3
            color =(255, 255, 255)
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 255), 4)  # Default width of 4
for regular buttons
        cv2.putText(img, button.text, (x + 20, y + 65), cv2.FONT_HERSHEY_SIMPLEX,
font_scale, color, boldness)
    return img
```



6.4 Pressing Key

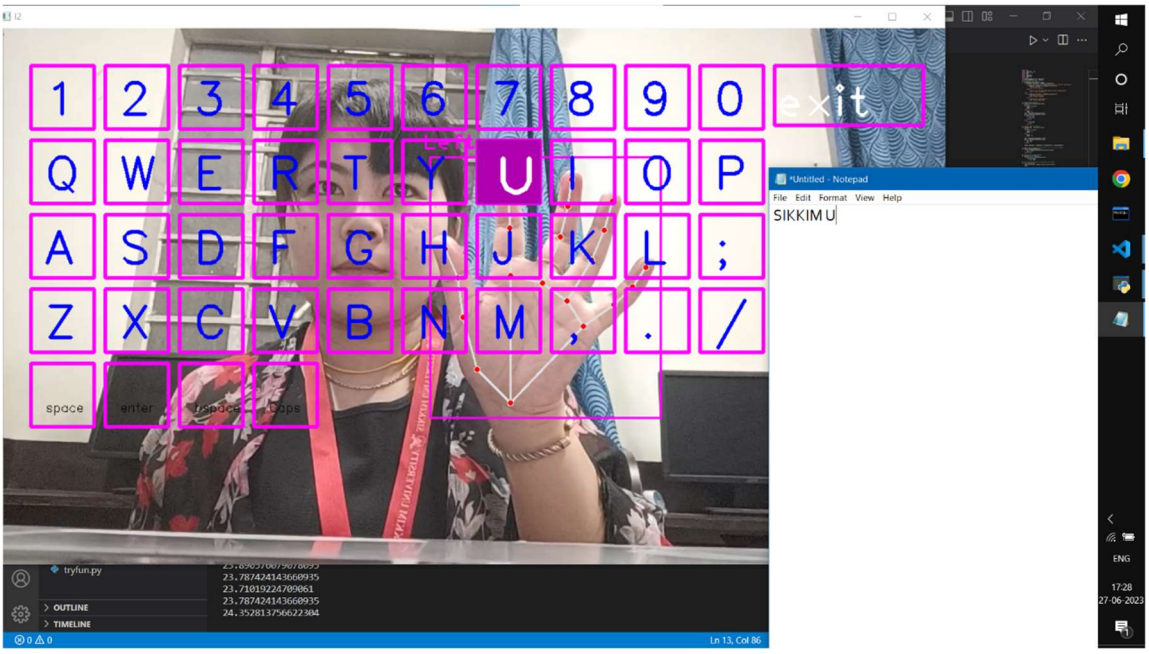
The Idea behind pressing button is we create a bound box over hand and we

calculate the bound box height and we calculate distance between index finger tip and index finger base landmark which is usually in case of straight hand 23 to 24 percent of the height of bound box . So for the pressing button when we hover over on any button and we reduce distance between index finger tip and index finger base to in between 20% to 17% by bending finger. When this condition is satisfy ,then we call pynput keyboard control press() function and we successfully press button.

```
x1, y1 = lmList1[8][0], lmList1[8][1]  # Landmark 8
                x2, y2 = lmList1[6][0], lmList1[6][1]  # Landmark 12

                length = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
                top_left = (bbox1[0], bbox1[1])
                bottom_left = (bbox1[0], bbox1[1] + bbox1[3])
                distance = math.sqrt((bottom_left[0] - top_left[0]) ** 2 +
(bottom_left[1] - top_left[1]) ** 2)
                percentage = (length * 100) / distance
                print(percentage)


                   keyboard.press(button.text)
                   sleep(0.40)
                   keyboard.release(button.text)
```



Chapter 7

Application Code:

main.py:

```
import cv2
import mediapipe as mp
import numpy as np
import time
import webbrowser
import subprocess
import math
from pynput.keyboard import Controller
from VirtualKeyboard import keyprint
# checking hand Orientation
def checkOrientation(landmarks_list1):
   if landmarks_list1[3][0] > landmarks_list1[17][0]:
      """ when x-coordinate of 3rd index(thumb)is greater than 17th index(pinky)"""
      if (landmarks_list1[0][1] > landmarks_list1[12][1]):
         """Upside orientate """
         return ("left","upside")
      else:
         """else it will be right hand properties which is upside down"""
         return ("right","downside")
```

```
    else:
        if (landmarks_list1[0][1] > landmarks_list1[12][1]):
            """#upright left hand 1"""
            return ("right","upside")
        else:
            """#upside down right hand 1"""
            return ("left","downside")

# checking thumb Orientation
def thumb(fing,coordinate,side,orient):
    axis =0
    if orient == "upside":
        upper = 0
        lower = 1
    else:
        upper = 1
        lower = 0
    tip = coordinate[fing[upper]][axis]
    base = coordinate[fing[lower]][axis]
    if side == "left":
        if tip > base:
            return 0
        return 1
    else:
        if tip < base:
            return 0
```

```
        return 1


# checking fingers Orientation
def checkUp(fing , coordinate,orient):
    axis = 1
    if orient == "upside":
        upper = 0
        lower = 1
    else:
        upper = 1
        lower = 0


    tip = coordinate[fing[upper]] [axis]
    base = coordinate[fing[lower]][axis]
    if tip > base:
        return 1
    return 0


    status_right[idx] = checkUp(val, coordinatelist, orient=handle[1])


# checking calling command is available or not in our list of command
def check_availableCommand(val):
    if val in command_function_identification.values():
        return True
    return False
```

```
# assigning a key to each command
def get_key(val, my_command):
    for key, value in my_command.items():
        if val == value:
            return key

# fire up the function that is associated with command
def fire_function(val):
    comm = get_key(val, command_function_identification)
    comm()



# defining commands
def command1():
    print("Opening YouTube")
    webbrowser.open('http://www.youtube.com')

def command2():
    print("Opening Notepad")
    subprocess.Popen("C:\\Windows\\notepad.exe")

def command3():
    print("Opening Calculator")
    subprocess.Popen("calc.exe")

def command4():
```

```
    print("Running virtual keyboard")
    keyprint()



# command dictionary
command_function_identification = {
    command1: [1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
    command2: [1, 1, 0, 0, 1, 0, 0, 0, 0, 0],
    command3: [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    command4: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
}

coordinate = {val: (0, 0) for val in range(0, 21)}
finger_val = [(4, 2), (8, 6), (12, 10), (16, 14), (20, 18)]
coordinatelist = {val: (0, 0) for val in range(0, 21)}
command_call = [[0], [0]]

keyboard_visible = False



cap = cv2.VideoCapture(0)
cap.set(3,1280)

mphands = mp.solutions.hands
hands = mphands.Hands(max_num_hands=2)
```

```
mp_drawing = mp.solutions.drawing_utils

while cap.isOpened():
   status_left = [0 for _ in finger_val]
   status_right = [0 for _ in finger_val]
   success, img = cap.read()
   height, width, channel = img.shape
   img = cv2.flip(img, 1)
   imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
   results = hands.process(imgRGB)

   if results.multi_hand_landmarks:
      for hand_landmarks in results.multi_hand_landmarks:
         mp_drawing.draw_landmarks(img, hand_landmarks,
mphands.HAND_CONNECTIONS)

         for id, lm in enumerate(hand_landmarks.landmark):
            h, w, c = img.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            coordinatelist[id] = (cx, cy)
         handle = checkOrientation(coordinatelist)

         if handle[0] == 'right':
            for idx, val in enumerate(finger_val):
               if idx == 0:
                  status_right[idx] = thumb(val, coordinatelist, side="right",
```

```
orient=handle[1])
                else:
                    status_right[idx] = checkUp(val, coordinatelist, orient=handle[1])
            elif handle[0] == 'left':
                for idx, val in enumerate(finger_val):
                    if idx == 0:
                        status_left[idx] = thumb(val, coordinatelist, side="left", orient=handle[1])
                    else:
                        status_left[idx] = checkUp(val, coordinatelist, orient=handle[1])
                status_left.reverse()

            cv2.putText(img, f'{handle}', coordinatelist[0], cv2.FONT_HERSHEY_PLAIN, 3,
(255, 0, 0), 2)

        cv2.putText(img, "Command " + str(status_left + status_right), (0, 400),
cv2.FONT_HERSHEY_PLAIN, 1.4, (255, 0, 0), 2)

        combicommand = status_left + status_right

        # accessing Command
        if combicommand != command_call[1]:
            command_call[1] = combicommand

            if command_call[0] != command_call[1]:
                print("both", command_call[0], command_call[1])
```

```
        if check_availableCommand(command_call[0]):
            fire_function(command_call[0])
        else:
            print("On this sign, we don't have an action")

        command_call[0] = command_call[1]
        command_call[1] = command_call[0]



cv2.imshow('I1', img)
if cv2.waitKey(1) == ord('s'):
    cv2.destroyWindow('I1')
    break
```

VirtualKeyboard.py:

```
import cv2
import math
from time import sleep
from pynput.keyboard import Controller


from cvzone.HandTrackingModule import HandDetector
def keyprint():
   cap = cv2.VideoCapture(0)
   cap.set(3, 1280)


   detector = HandDetector(detectionCon=0.8, maxHands=2)


   keys = [["1", "2", "3", "4", "5", "6", "7", "8", "9", "0"],
         ["Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P"],
         ["A", "S", "D", "F", "G", "H", "J", "K", "L", ";"],
         ["Z", "X", "C", "V", "B", "N", "M", ",", ".", "/"],
         ["space", "enter", "bspace","Caps"]
         ]
   def change_keys(keys):
      if keys == [["1", "2", "3", "4", "5", "6", "7", "8", "9", "0"],
            ["Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P"],
            ["A", "S", "D", "F", "G", "H", "J", "K", "L", ";"],
            ["Z", "X", "C", "V", "B", "N", "M", ",", ".", "/"],
```

```
            ["space", "enter", "bspace", "Caps"]
        ]:
        keys = [["1", "2", "3", "4", "5", "6", "7", "8", "9", "0"],
            ["q", "w", "e", "r", "t", "y", "u", "i", "o", "p"],
            ["a", "s", "d", "f", "g", "h", "j", "k", "l", ";"],
            ["z", "x", "c", "v", "b", "n", "m", ",", ".", "/"],
            ["space", "enter", "bspace", "Caps"]
            ]
        return keys
    else:
        keys = [["1", "2", "3", "4", "5", "6", "7", "8", "9", "0"],
            ["Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P"],
            ["A", "S", "D", "F", "G", "H", "J", "K", "L", ";"],
            ["Z", "X", "C", "V", "B", "N", "M", ",", ".", "/"],
            ["space", "enter", "bspace", "Caps"]
            ]
        return keys


keyboard = Controller()
def drawAll(img, buttonList):
    for i, button in enumerate(buttonList):
        x, y = button.pos
        w, h = button.size
        if button.text in ["space", "enter", "bspace","Caps"]:
            font_scale = .56
            boldness = 1
```

```
            color =(0, 0, 0)
        else:
            font_scale = 2.0
            boldness = 3
            color =(255, 0, 0)
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 255), 4)
        cv2.putText(img, button.text, (x + 20, y + 65), cv2.FONT_HERSHEY_SIMPLEX,
font_scale, color, boldness)
    return img

class Button():
    def __init__(self, pos, text, size=[85, 85]):
        self.pos = pos
        self.size = size
        self.text = text

def myexit():
    cv2.destroyWindow('I2')

buttonList = []
for i in range(len(keys)):
    for j, key in enumerate(keys[i]):
        buttonList.append(Button([100 * j + 50, 100 * i + 50], key))

while cap.isOpened():
    success, img = cap.read()
```

```
        img = cv2.flip(img, 1)
        hands, img = detector.findHands(img)
        img = drawAll(img, buttonList)

        cv2.rectangle(img, (1050, 50), (1250, 130), (255, 0, 255), 4)
        cv2.putText(img, "exit", (1050, 120), cv2.FONT_HERSHEY_PLAIN, 4, (255, 255, 255),
4)

        if hands:
            hand1 = hands[0]
            lmList1 = hand1["lmList"]
            bbox1 = hand1["bbox"]
            centerPoint1 = hand1["center"]
            handType = hand1["type"]

            if lmList1:
                for button in buttonList:
                    x, y = button.pos
                    w, h = button.size

                    if x < lmList1[8][0] < x + w and y < lmList1[8][1] < y + h:
                        cv2.rectangle(img, button.pos, (x + w, y + h), (175, 0, 175), cv2.FILLED)
                        cv2.putText(img, button.text, (x + 25, y + 75), cv2.FONT_HERSHEY_PLAIN,
5, (255, 255, 255), 5)

                        x1, y1 = lmList1[8][0], lmList1[8][1]  # Landmark 8
```

```
                x2, y2 = lmList1[6][0], lmList1[6][1]  # Landmark 12

                length = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
                top_left = (bbox1[0], bbox1[1])
                bottom_left = (bbox1[0], bbox1[1] + bbox1[3])
                distance = math.sqrt((bottom_left[0] - top_left[0]) ** 2 + (bottom_left[1] -
top_left[1]) ** 2)
                percentage = (length * 100) / distance
                print(percentage)

                if 17 <= percentage < 20:
                   if button.text == "enter":
                      keyboard.press('\n')
                      sleep(0.40)
                      keyboard.release('\n')
                   elif button.text == "bspace":
                      keyboard.press('\b')
                      sleep(0.40)
                      keyboard.release('\b')
                   elif button.text == "space":
                      keyboard.press(' ')
                      sleep(0.40)
                      keyboard.release(' ')
                   elif button.text == "Caps":
                      keys = change_keys(keys)
                      buttonList = []
```

```
                    for i in range(len(keys)):
                        for j, key in enumerate(keys[i]):
                            buttonList.append(Button([100 * j + 50, 100 * i + 50], key))
                    sleep(0.40)
                else:
                    keyboard.press(button.text)
                    sleep(0.40)
                    keyboard.release(button.text)

            elif 1050 < lmList1[8][0] < 1250 and 50 < lmList1[8][1] < 130:
                cv2.rectangle(img, (1050, 50), (1250, 130), (255, 0, 255), cv2.FILLED)
                cv2.putText(img, "exit", (1050, 120), cv2.FONT_HERSHEY_PLAIN, 4, (255,
255, 255), 4)

                x1, y1 = lmList1[8][0], lmList1[8][1]  # Landmark 8
                x2, y2 = lmList1[6][0], lmList1[6][1]  # Landmark 12

                length = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
                top_left = (bbox1[0], bbox1[1])
                bottom_left = (bbox1[0], bbox1[1] + bbox1[3])
                distance = math.sqrt((bottom_left[0] - top_left[0]) ** 2 + (bottom_left[1] -
top_left[1]) ** 2)
                percentage = (length * 100) / distance
                print(percentage)

                if 17 <= percentage < 20:
```

```
                myexit()


cv2.imshow('I2', img)
if cv2.waitKey(1) & 0xFF == ord('r'):
   cv2.destroyWindow('I2')
   break
```

Chapter 8

Conclusion

Human interaction with devices will keep on increasing at a rapid rate. The number of devices a person interacts with in a day is much more as compared to 20 years ago, and this will keep on increasing. Technology has become a part of our everyday work and will do for years to come. Seeing the number of devices, one interacts with and the limited physical ability of a human structure new ways of interacting with smart devices will come up significantly in the future. With our project we have tried to test out one such way.
This work aims to design methodologies to enable contact-free interaction with computer systems using the human hand gesture and initiate various control actions. We have shown how we can create a contact-free interaction with a device with the hardware that it already inherits. There has been a significant progressin the Hand gesture recognition.
With deep learning and AI there has been a significant improvement in accuracy, which has led to the development of specialized models.

7.1 Limitations

With this project we were able to show how hand gestures can work as a command to control device. Although we have fulfilled the object set in the beginning of this project there are still some limitations. In an ideal situation we are able to give out the command but there are some anomalous situations such as more than a pair hands in the image frame could decrease the accuracy and cause wrong data to be interpreted. There are also some situations when the orientation detection could fail such as when the y-axis of the landmark 12 is equal to that of landmark 0.

Platform Compatibility: The code has dependencies on external libraries like OpenCV and pynput. It may require specific installations and configurations to work correctly on different platforms or operating systems. Ensuring compatibility across various platforms may require additional testing and adjustments.

Performance: The performance of the virtual keyboard may be affected by factors such as the processing power of the system, camera resolution, and frame rate. In resource-constrained environments or with low-quality webcams, there might be

noticeable delays or lag between hand movements and keyboard respons

7.1 Further work

- Create a GUI for users to let them create custom commands.
- Give users the ability to customize the actions that the commands execute.
- Integrate the control actions of a mouse.
- Increase the overall accuracy.

We took up the objective to- create a contact-free interaction using hand gestures and were able to achieve the goals that we had set. There are some places that need to be improvised and beautify the application so we will continue to develop and improvise it.

Bibliography

[1] Rafiqul Zaman Khan, Noor Ibraheem, (2012). Hand Gesture Recognition. *International Journal of Artificial Intelligence Applications 3(4):161-174*. https://tinyurl.com/handGestureRecognition

[2] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. (2015). SSD: Single Shot MultiBox Detector. https://arxiv.org/abs/1512.02325

[3] Hands. mediapipe. (n.d.). Retrieved July 15, 2022, from https://google.github.io/mediapipe/solutions/hands.html

[4] Cvzone, https://github.com/cvzone/cvzone