

Gesture Control Advancements

A PROJECT REPORT

Submitted By

SRISHTY SINGH

(2100290140136)

RAJ PRATAP SINGH

(2100290140109)

**Submitted in partial fulfilment of the
Requirements for the Degree of**

MASTER OF COMPUTER APPLICATION

**Under the Supervision of
Mr. Prashant Agrawal
Associate Professor**



Submitted to

**DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206**

(02 JUNE 2023)

CERTIFICATE

Certified that **Srishty Singh (2100290140136), Raj Pratap Singh (2100290140109)** have carried out the project work having “**Hand Gestures Recognition Using Mediapipe**” for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the students themselves and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date: **Srishty Singh (2100290140136)**
Raj Pratap Singh (2100290140109)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: **Mr. Prashant Agrawal**
Associate Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Signature of Internal Examiner

Signature of External Examiner

Dr. Arun Tripathi
Head, Department of Computer Applications
KIET Group of Institutions, Ghaziabad

ABSTRACT

The project performs Gesture Control, making use of Google's Mediapipe framework in python, without any physical aid being affixed to hands.

MediaPipe offers customizable ML solutions. Mediapipe Hands is a hand and finger tracking solution. It works by deducing 21 3D landmarks of a hand. For which the pipeline consists of two modules: a palm detection model after that a hand landmark model. The palm detector achieves an average precision of 95.7%. The hand detector performs key-point localization of all 21 hand-knuckle coordinates in the detected hand regions provided by palm detector employing regression.

The project captures live video feed from camera using opencv library then pass it to Mediapipe that returns the landmarks' details. Afterwards it operate on those landmarks to implement several trivial use cases. By computing relative positioning, distance among landmarks, linear alignment we implement virtual mouse, finger counter, volume controller, rock-paper-scissors game. The required GUI is brought about using opencv.

ACKNOWLEDGEMENT

Success in life is never attained single handedly. My deepest gratitude goes to my thesis supervisor, **Mr. Prashant Agrawal** for his guidance, help and encouragement throughout my research work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Arun Kumar Tripathi, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Srishty Singh (2100290140136)

Raj Pratap Singh (2100290140109)

TABLE OF CONTENTS

Certificate	i
Abstract	ii
Acknowledgement	iii
Table of Contents	iv
1. Introduction	6-9
1.1 Project Description	6-7
1.2 Project Methodology	8
1.3 Hardware / Software used	9
2 Feasibility Study	10
2.1 Technical Feasibility	10
2.2 Operational Feasibility	10
2.3 Economical Feasibility	10
3 Design (Input/Output Screenshots)	11-16
3.1 Hand Tracking	11
3.2 Finger Counter	12
3.3 Virtual Mouse Pointer	13
3.4 Volume Controller	14
3.5 Rock-Paper-Scissors Game	15
3.6 Snake Game	16
4 Coding	17-31
5 Comparison	32
6 Limitations	33
7 Future Scope	34
8 Conclusion	35
9 Bibliography	36-38

INTRODUCTION

1.1 PROJECT DESCRIPTION

What is Gesture Recognition?

Gesture recognition is the mathematical interpretation of a human hand motion by a computing device. It is a sub-discipline of computer vision.

Gesture recognition is a type of perceptual computing user interface that allows computers to capture and interpret human gestures as commands. The general definition of gesture recognition is the ability of a computer to understand gestures and execute commands based on those gestures. Users can make simple gestures to control or interact with devices without physically touching them.

Gesture recognition can be seen as a way for computers to begin to understand human body language, thus building a better bridge between machines and humans than older text user interfaces or even GUIs (graphical user interfaces), which still limit the majority of input to keyboard and mouse and interact naturally without any mechanical devices.

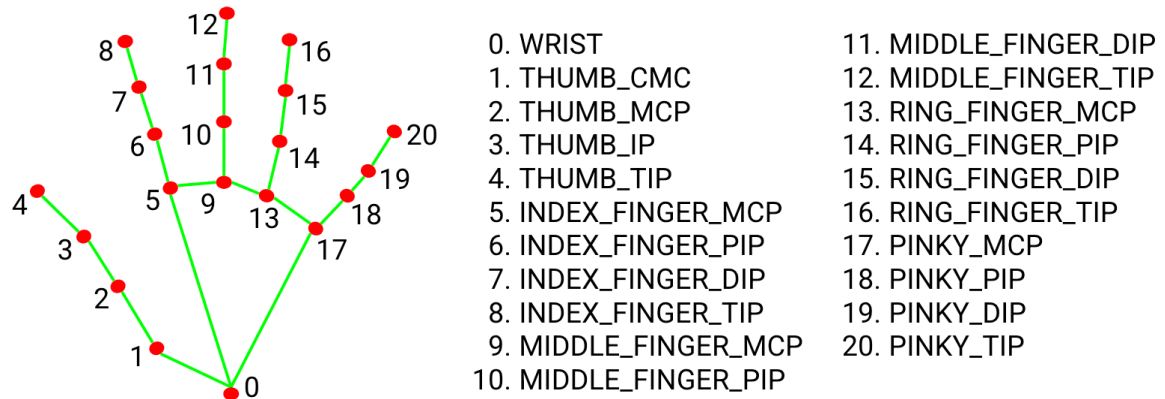
What is Mediapipe Hands?

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. Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, this method achieves real-time performance on a mobile phone, and even scales to multiple hands.

MediaPipe Hands utilizes an ML pipeline consisting of multiple models working together: A palm detection model that operates on the full image and returns an oriented hand bounding box. A hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand keypoints.

Palm Detection Model : This trains a palm detector instead of a hand detector, since estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. It achieves an average precision of 95.7% in palm detection.

Hand Landmark Model : After the palm detection over the whole image our subsequent hand landmark model performs precise keypoint localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, that is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.



Implementation

Afterwards operating on the landmarks provided by the model, project implements several trivial use cases, by computing relative positioning, distance among landmarks, linear alignment.

Applications implemented in the project are:

- Virtual mouse pointer with left click operation.
- Volume controller
- Finger counter for 0-5 in right-hand
- Rock-Paper-Scissors game with AI as opponent
- Snake game

1.2 PROJECT METHODOLOGY

- 1- MediaPipe-** MediaPipe is an artificial intelligence framework provided by Google. This is a service that provides a solution-type library so that the human body recognition function model included in the image data can be developed and learned and used easily. MediaPipe's representative solutions are configured as shown in Table 2, and continue to provide new solutions. MediaPipe supports various development environments such as web pages, Android, and iOS. And supported languages include Android, iOS, C++, Python, JS, and Coral. In addition, the media pipe is an opensource project, and the program source is disclosed, so you can modify what you want and use it for development. In this study, the Hands solution, which recognizes the shape and movement of the hand among the MediaPipe solutions, was used.
- 2- Mediapipe and Hand tracking-** MediaPipe hand tracking is a model that uses a machine learning pipeline to detect and track recognized hands and fingers in images. After detecting the palm of the entire image using machine learning, as shown in Figure 1, the method uses a method of mapping 21 coordinates to define 3D coordinates to display the joints of the palm and obtain real data. MediaPipe hand tracking provides real-time performance that Han et al, Journal of System and Management Sciences, Vol. 12 (2022) No. 2, pp. 462-476 467 can be used not only on desktop computers but also on smartphones, and can be expanded to recognize multiple hands at once.
- 3- ML Pipeline-** MediaPipe Hands utilizes a multi-model ML pipeline. Representative examples are the palm detection model, which operates on the entire image and returns an oriented hand bounding box, and the hand landmark model, which operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand keypoints.

1.3 HARDWARE / SOFTWARE USED

Hardware Used :

1. Mainframe - Intel core i5 ninth generation.
2. Python put in Windows Os or Linux Os Machine
3. GPU - Nvidia GTX 1050 Ti.
4. 720p60 net Camera

Software Used :

1. Windows Os or Linux Os
2. Python 3
3. Open-Cv
4. Mediapipe
5. Tensorflow
6. Pycharm Code Editor

FEASIBILITY STUDY

2.1 Technical Feasibility

- The basic hardware requirement of this project is a camera which can be easily found in any system or availed individually.
- It doesn't require high processing power so it can run on all machines.
- The basic software requirement of this project is some libraries which can be easily installed from the internet.

The above points conclude that the project is technically feasible to be developed.

2.2 Operational Feasibility

- The main purpose of building this project is to minimize the usage of input hardware like mouse and keyboard to interact with the system.
- It provides an innovative idea to use hand gestures to interact with the system instead of input devices.

The above points concludes that the project is operationally feasible to be developed.

2.3 Economical Feasibility

- The basic hardware requirement of this project is a camera which can be easily found in any system. So, there is no cost requirement for any external hardware..
- It doesn't require high processing power so it can run on all machines.
- The basic software requirement of this project is some libraries which can be easily installed from the internet. Those libraries are free of cost.

The above points concludes that the project doesn't require any extra cost to be developed. Hence, It is economically feasible.

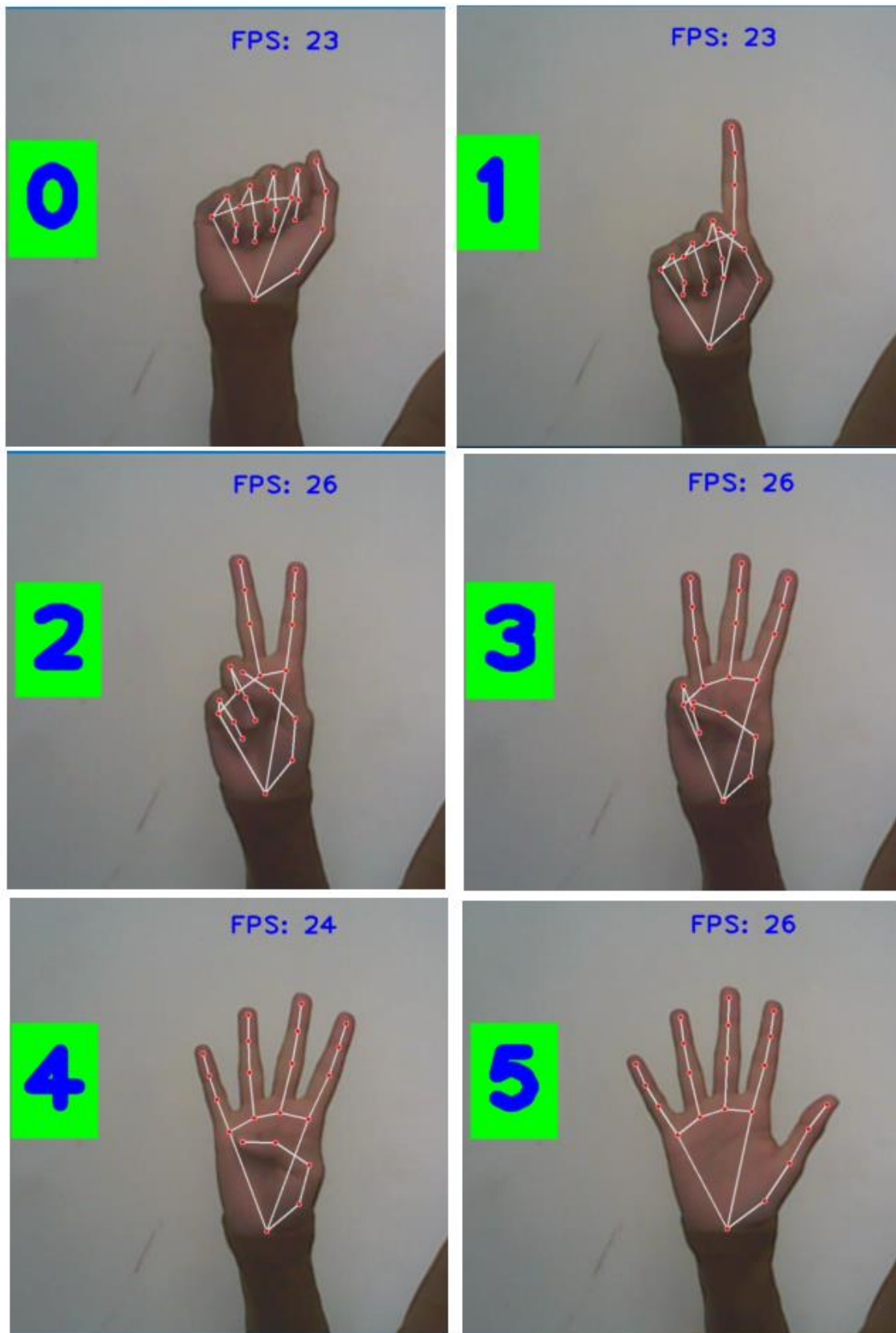
DESIGN

3.1 Hand Tracking

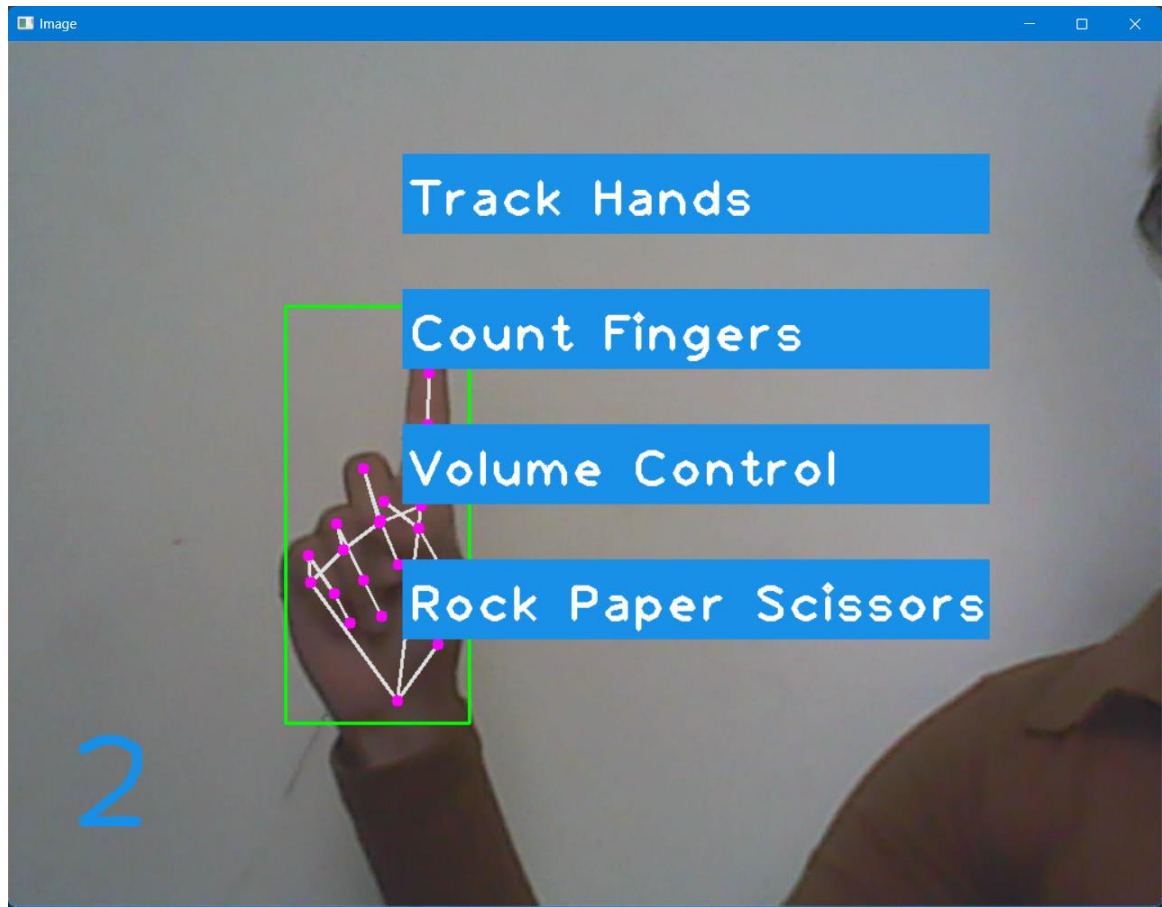


Hand tracking is the process of detecting and tracking the movements and gestures of the human hand in real-time using camera feed. Hand tracking has many applications in fields such as virtual reality, gaming, robotics, human-computer interaction, and sign language recognition.

3.2 Finger Counter

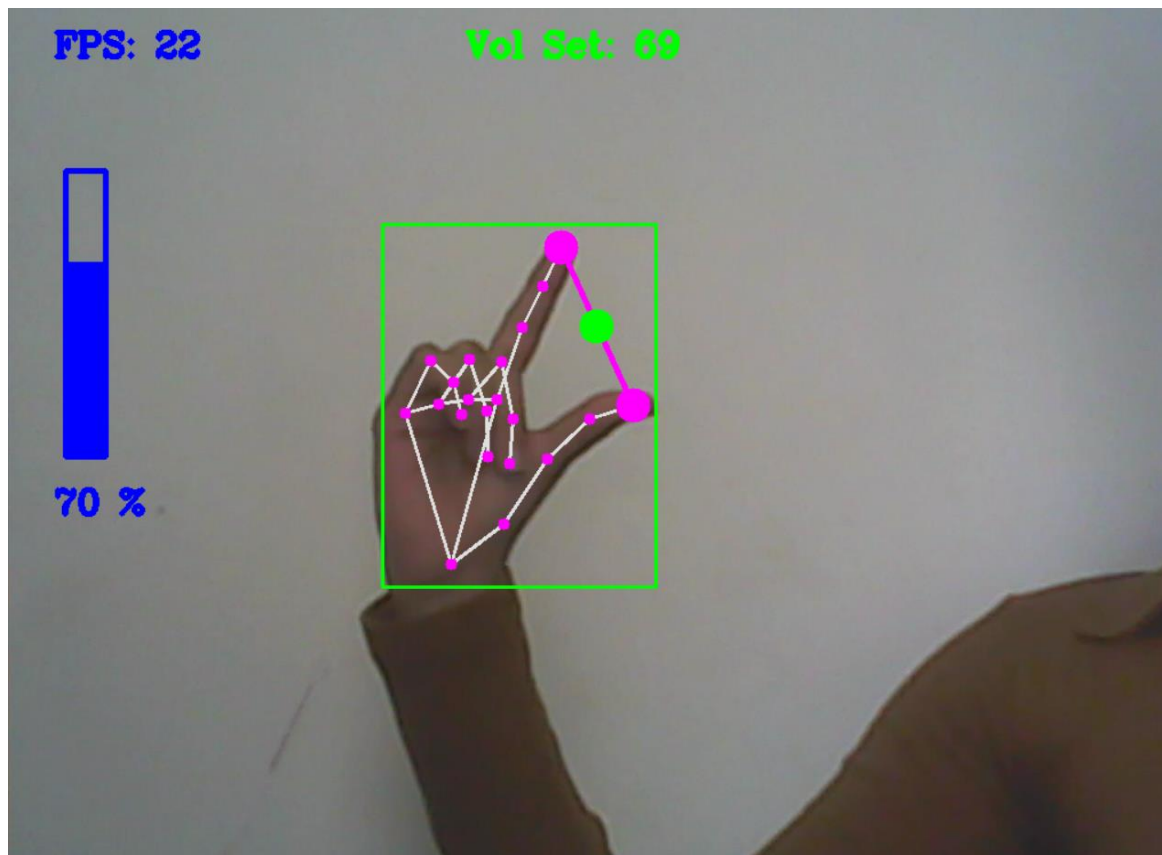


3.3 Virtual Mouse Pointer



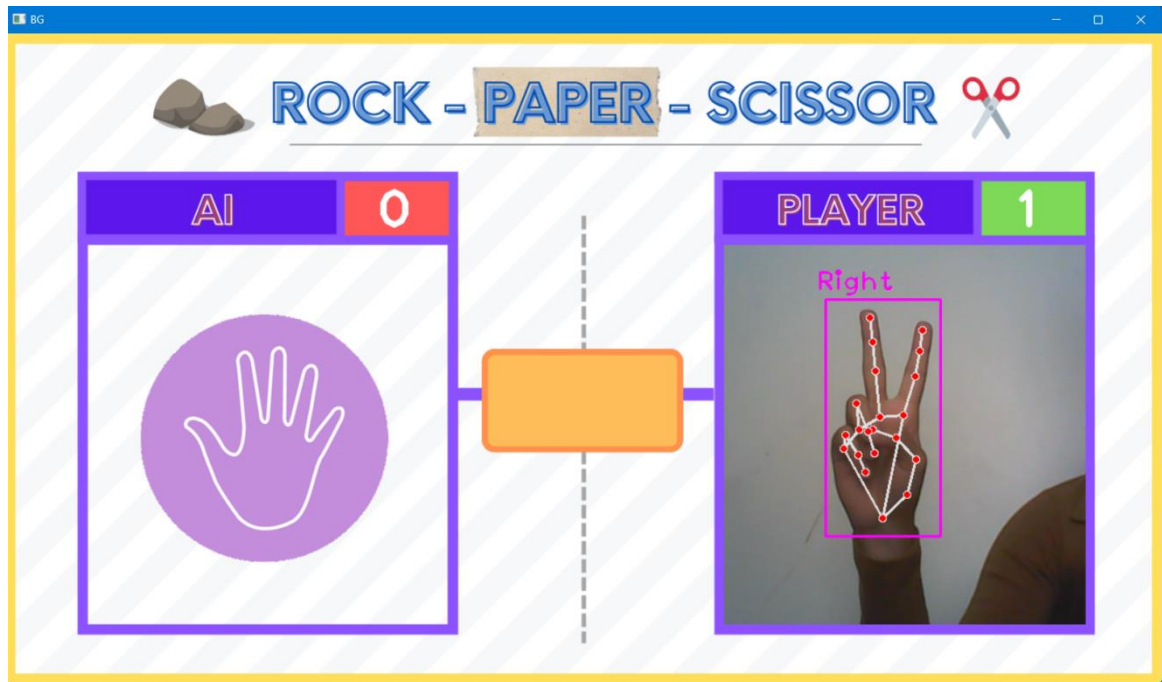
A virtual mouse pointer enables users to control the movement of the mouse pointer on their computer screen without using a physical mouse. Virtual mouse pointers are particularly useful for individuals who have mobility impairments or who prefer alternative input devices to a traditional mouse.

3.4 Volume Controller



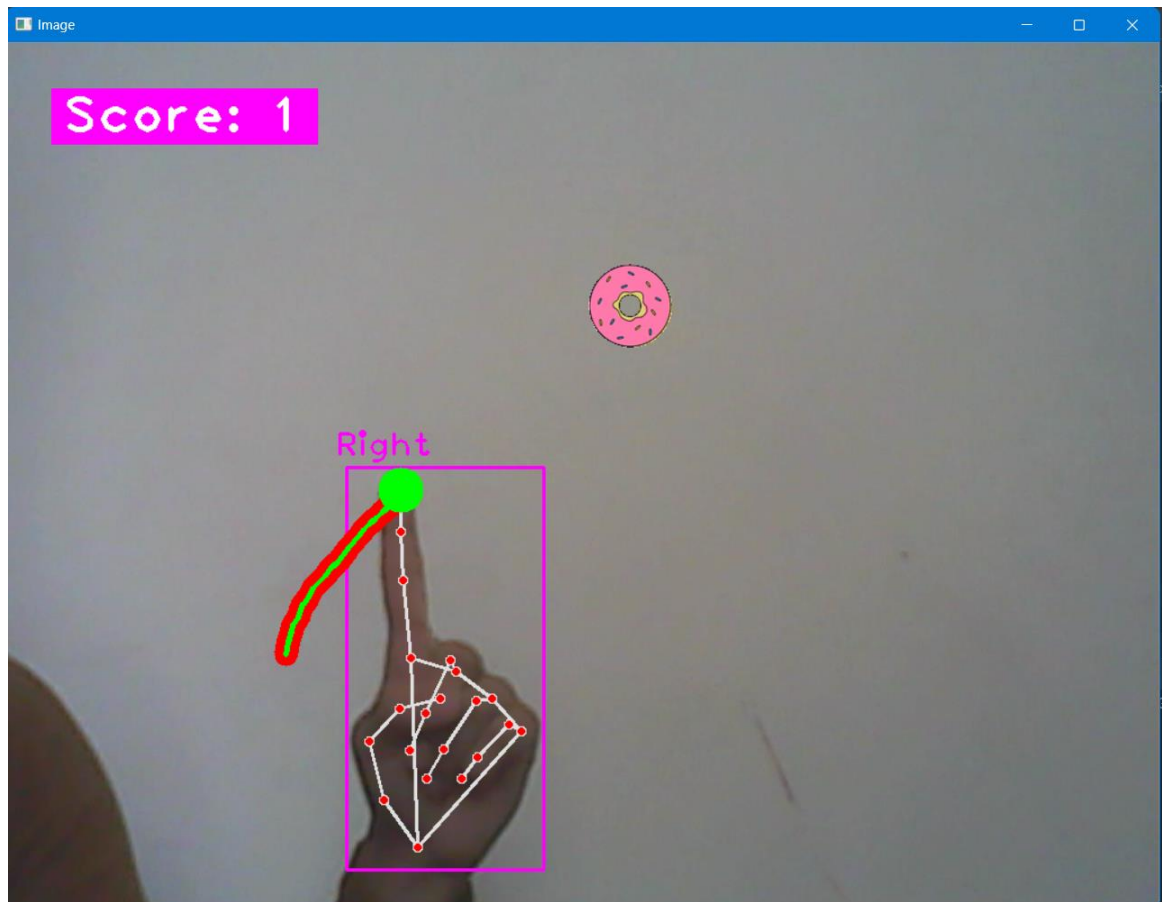
Volume controller enables users to control their system's output volume using simple hand gestures. Volume control using hand gestures is particularly useful in situations where users may not have direct access to their devices, such as when listening to music or watching a video from a distance, or when their hands are occupied with other tasks.

3.5 Rock-Paper-Scissors Game



User can also play some games like Rock-Paper-Scissors and Snake game using with simple hand gestures. Both the games are interactive. Many more games can be played using hand gesture which enables hand gesture recognition to be a very innovative technology in the field of game development.

3.6 Snake Game



CODING

4.1 HandTrackingModule.py

```
import cv2
import mediapipe as mp
import time
import math

class HandDetector():
    def __init__(self, mode=False, maxHands=2, model_complexity=1, detectionCon=0.5,
trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.trackCon = trackCon
        self.model_complexity = model_complexity

        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands, self.model_complexity,
self.detectionCon, self.trackCon)
        self.mpDraw = mp.solutions.drawing_utils
        self.tipIds = [4, 8, 12, 16, 20]

    def findHands(self, img, draw=True):
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
        # print(results.multi_hand_landmarks)

        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms, self.mpHands.HAND_CON-
NECTIONS)
            return img

    def findPositionVol(self, img, handNo=0, draw=True):
        xList = []
        yList = []
        bbox = []
        self.lmList = []
        if self.results.multi_hand_landmarks:
            myHand = self.results.multi_hand_landmarks[handNo]
            for id, lm in enumerate(myHand.landmark):
                # print(id, lm)
                h, w, c = img.shape
```

```

        cx, cy = int(lm.x * w), int(lm.y * h)
        xList.append(cx)
        yList.append(cy)
        # print(id, cx, cy)
        self.lmList.append([id, cx, cy])
        if draw:
            cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)
        xmin, xmax = min(xList), max(xList)
        ymin, ymax = min(yList), max(yList)
        bbox = xmin, ymin, xmax, ymax

    if draw:
        cv2.rectangle(img, (bbox[0] - 20, bbox[1] - 20),
                      (bbox[2] + 20, bbox[3] + 20), (0, 255, 0), 2)

    return self.lmList, bbox

def findPosition(self, img, handNo=0, draw=True):

    lmList = []
    if self.results.multi_hand_landmarks:
        myHand = self.results.multi_hand_landmarks[handNo]
        for id, lm in enumerate(myHand.landmark):
            # print(id, lm)
            h, w, c = img.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            # print(id, cx, cy)
            lmList.append([id, cx, cy])
            if draw:
                cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)

    return lmList

def fingersUp(self):
    fingers = []
    # Thumb
    if self.lmList[self.tipIds[0]][1] > self.lmList[self.tipIds[0] - 1][1]:
        fingers.append(1)
    else:
        fingers.append(0)
    # 4 Fingers
    for id in range(1, 5):
        if self.lmList[self.tipIds[id]][2] < self.lmList[self.tipIds[id] - 2][2]:
            fingers.append(1)
        else:
            fingers.append(0)
    return fingers

```

```

def findDistance(self, p1, p2, img, draw=True):
    x1, y1 = self.lmList[p1][1], self.lmList[p1][2]
    x2, y2 = self.lmList[p2][1], self.lmList[p2][2]
    cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

    if draw:
        cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
        cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
        cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
        cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)

    length = math.hypot(x2 - x1, y2 - y1)
    return length, img, [x1, y1, x2, y2, cx, cy]

def main():
    pTime = 0
    cap = cv2.VideoCapture(1)
    detector = HandDetector()
    while True:
        success, img = cap.read()
        img = detector.findHands(img)
        lmList = detector.findPosition(img)
        if len(lmList) != 0:
            print(lmList[4])

        cTime = time.time()
        fps = 1 / (cTime - pTime)
        pTime = cTime

        cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3,
                    (255, 0, 255), 3)

        cv2.imshow("Image", img)
        cv2.waitKey(1)

if __name__ == "__main__":
    main()

```

4.2 FingerCounter.py

```
import cv2
import time
import os

import HandTrackingModule
import HandTrackingModule as htm

wCam, hCam = 1920, 1080

cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)

pTime = 0

detector = htm.HandDetector(detectionCon=0.75, maxHands=1)

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

while True:
    success, img = cap.read()
    img = detector.findHands(img)
    lmList = detector.findPosition(img, draw=False)
    print(lmList)

    if len(lmList) != 0:
        fingers = []




        # Thumb
        if lmList[tipIds[0]][1] > lmList[tipIds[0] - 1][1]:
            fingers.append(1)
        else:
            fingers.append(0)

        # 4 Fingers
        for id in range(1, 5):
            if lmList[tipIds[id]][2] < lmList[tipIds[id] - 2][2]:
                fingers.append(1)
            else:
                fingers.append(0)



        # print(fingers)
        totalFingers = fingers.count(1)
        print(totalFingers)

        #h, w, c = overlayList[totalFingers - 1].shape
```

```

cTime = time.time()
fps = 1 / (cTime - pTime)
pTime = cTime

cv2.imshow("Image", img)
cv2.waitKey(1)

```

4.3 VirtualMouseMenu.py

```
import cv2
from HandTrackingModule import HandDetector
from time import sleep
import numpy as np
import cvzone
from pynput.keyboard import Controller

cap = cv2.VideoCapture(0)
cap.set(3, 1920)
cap.set(4, 1080)

detector = HandDetector(detectionCon=0.8)
keys = [
    ["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", ",", ".", "/"]
]
finalText = ""

keyboard = Controller()

def drawAll(img, buttonList):
    for button in buttonList:
        x, y = button.pos
        w, h = button.size
        cvzone.cornerRect(img, (button.pos[0], button.pos[1], button.size[0], button.size[1]),
                           20, rt=0)
        cv2.rectangle(img, button.pos, (x + w, y + h), (255, 0, 255), cv2.FILLED)
        cv2.putText(img, button.text, (x + 10, y + 55),
                    cv2.FONT_HERSHEY_PLAIN, 4, (255, 255, 255), 4)
    return img
```

```

class Button():
    def __init__(self, pos, text, size=[65, 65]):
        self.pos = pos
        self.size = size
        self.text = text

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 True:
    success, img = cap.read()
    img = detector.findHands(img)
    lmList, bboxInfo = detector.findPositionVol(img)
    img = drawAll(img, buttonList)

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

            if x < lmList[8][0] < x + w and y < lmList[8][1] < y + h:
                cv2.rectangle(img, (x - 5, y - 5), (x + w + 5, y + h + 5), (175, 0, 175),
cv2.FILLED)
                cv2.putText(img, button.text, (x + 20, y + 65),
                    cv2.FONT_HERSHEY_PLAIN, 4, (255, 255, 255), 4)
                l, _, _ = detector.findDistance(8, 12, img, draw=False)
                print(l)

                ## when clicked
                if l < 30:

```

```
keyboard.press(button.text)
cv2.rectangle(img, button.pos, (x + w, y + h), (0, 255, 0), cv2.FILLED)
cv2.putText(img, button.text, (x + 20, y + 65),
            cv2.FONT_HERSHEY_PLAIN, 4, (255, 255, 255), 4)
finalText += button.text
sleep(0.15)
```

```
cv2.rectangle(img, (50, 350), (700, 450), (175, 0, 175), cv2.FILLED)
cv2.putText(img, finalText, (60, 430),
            cv2.FONT_HERSHEY_PLAIN, 5, (255, 255, 255), 5)
cv2.imshow("Image", img)
cv2.waitKey(1)
```


4.4 VolumeController.py

```
import cv2
import time
import numpy as np
import HandTrackingModule as htm
import math
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

#####
wCam, hCam = 1920, 1080
#####

cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
pTime = 0

detector = htm.HandDetector(detectionCon=0.7, maxHands=1)

devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(
    IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
# volume.GetMute()
# volume.GetMasterVolumeLevel()
volRange = volume.GetVolumeRange()
minVol = volRange[0]
maxVol = volRange[1]
vol = 0
volBar = 400
volPer = 0
area = 0
colorVol = (255, 0, 0)

while True:
    success, img = cap.read()

    # Find Hand
    img = detector.findHands(img)
    lmList, bbox = detector.findPositionVol(img, draw=True)
    if len(lmList) != 0:

        # Filter based on size
        area = (bbox[0] * bbox[1]) // 100
        # print(area)
```

```

if 250 < area < 1000:

    # Find Distance between index and Thumb
    length, img, lineInfo = detector.findDistance(4, 8, img)
    # print(length)

    # Convert Volume
    volBar = np.interp(length, [50, 200], [400, 150])
    volPer = np.interp(length, [50, 200], [0, 100])

    # Reduce Resolution to make it smoother
    smoothness = 10
    volPer = smoothness * round(volPer / smoothness)

    # Check fingers up
    fingers = detector.fingersUp()
    # print(fingers)

    # If pinky is down set volume
    if not fingers[4]:
        volume.SetMasterVolumeLevelScalar(volPer / 100, None)
        cv2.circle(img, (lineInfo[4], lineInfo[5]), 15, (0, 255, 0), cv2.FILLED)
        colorVol = (0, 255, 0)
    else:
        colorVol = (255, 0, 0)

    # Drawings
    cv2.rectangle(img, (50, 150), (85, 400), (255, 0, 0), 3)
    cv2.rectangle(img, (50, int(volBar)), (85, 400), (255, 0, 0), cv2.FILLED)
    cv2.putText(img, f'{int(volPer)} %', (40, 450), cv2.FONT_HERSHEY_COMPLEX,
                1, (255, 0, 0), 3)
    cVol = int(volume.GetMasterVolumeLevelScalar() * 100)
    cv2.putText(img, f'Vol Set: {int(cVol)}', (400, 50), cv2.FONT_HERSHEY_COMPLEX,
                1, colorVol, 3)

    # Frame rate
    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    cv2.putText(img, f'FPS: {int(fps)}', (40, 50), cv2.FONT_HERSHEY_COMPLEX,
                1, (255, 0, 0), 3)

    cv2.imshow("Img", img)
    cv2.waitKey(1)

```

4.5 RockPaperScissorsGame.py

```
import random
import cv2
import cvzone
from cvzone.HandTrackingModule import HandDetector
import time

cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

detector = HandDetector(maxHands=1)

timer = 1
stateResult = False
startGame = False
scores = [0, 0] # [AI, Player]

while True:
    imgBG = cv2.imread("Resources/BG.png")
    success, img = cap.read()

    imgScaled = cv2.resize(img, (0, 0), None, 0.875, 0.875)
    imgScaled = imgScaled[:, 80:480]

    # Find Hands
    hands, img = detector.findHands(imgScaled) # with draw

    if startGame:

        if stateResult is False:
            timer = time.time() - initialTime
            cv2.putText(imgBG, str(int(timer)), (605, 435), cv2.FONT_HERSHEY_PLAIN,
6, (255, 0, 255), 4)

            if timer > 3:
                stateResult = True
                timer = 1

            if hands:
                playerMove = None
                hand = hands[0]
                fingers = detector.fingersUp(hand)
                if fingers == [0, 0, 0, 0, 0]:
                    playerMove = 1
                if fingers == [1, 1, 1, 1, 1]:
                    playerMove = 2
```

```

    if fingers == [0, 1, 1, 0, 0]:
        playerMove = 3

    randomNumber = random.randint(1, 3)
    imgAI = cv2.imread(f'Resources/{randomNumber}.png', cv2.IMREAD_UN-
CHANGED)
    imgBG = cvzone.overlayPNG(imgBG, imgAI, (149, 310))

    # Player Wins
    if (playerMove == 1 and randomNumber == 3) or \
        (playerMove == 2 and randomNumber == 1) or \
        (playerMove == 3 and randomNumber == 2):
        scores[1] += 1

    # AI Wins
    if (playerMove == 3 and randomNumber == 1) or \
        (playerMove == 1 and randomNumber == 2) or \
        (playerMove == 2 and randomNumber == 3):
        scores[0] += 1

    imgBG[234:654, 795:1195] = imgScaled

    if stateResult:
        imgBG = cvzone.overlayPNG(imgBG, imgAI, (149, 310))

    cv2.putText(imgBG, str(scores[0]), (410, 215), cv2.FONT_HERSHEY_PLAIN, 4,
(255, 255, 255), 6)
    cv2.putText(imgBG, str(scores[1]), (1112, 215), cv2.FONT_HERSHEY_PLAIN, 4,
(255, 255, 255), 6)

    # cv2.imshow("Image", img)
    cv2.imshow("BG", imgBG)
    # cv2.imshow("Scaled", imgScaled)

    key = cv2.waitKey(1)
    if key == ord('s'):
        startGame = True
        initialTime = time.time()
        stateResult = False

```

4.6 SnakeGame.py

```
import math
import random
import cvzone
import cv2
import numpy as np
from cvzone.HandTrackingModule import HandDetector

cap = cv2.VideoCapture(0)
cap.set(3, 1920)
cap.set(4, 1080)

detector = HandDetector(detectionCon=0.8, maxHands=1)

class SnakeGameClass:
    def __init__(self, pathFood):
        self.points = [] # all points of the snake
        self.lengths = [] # distance between each point
        self.currentLength = 0 # total length of the snake
        self.allowedLength = 150 # total allowed Length
        self.previousHead = 0, 0 # previous head point

        self.imgFood = cv2.imread(pathFood, cv2.IMREAD_UNCHANGED)
        self.hFood, self.wFood, _ = self.imgFood.shape
        self.foodPoint = 0, 0
        self.randomFoodLocation()

        self.score = 0
        self.gameOver = False

    def randomFoodLocation(self):
        self.foodPoint = random.randint(100, 600), random.randint(100, 400)

    def update(self, imgMain, currentHead):
        if self.gameOver:
            cvzone.putTextRect(imgMain, "Game Over", [100, 400],
                               scale=3, thickness=2, offset=20)
            cvzone.putTextRect(imgMain, f'Your Score: {self.score}', [100, 350],
                               scale=3, thickness=2, offset=20)
            self.score = 0
        else:
            px, py = self.previousHead
            cx, cy = currentHead

            self.points.append([cx, cy])
```

```

distance = math.hypot(cx - px, cy - py)
self.lengths.append(distance)
self.currentLength += distance
self.previousHead = cx, cy

# Length Reduction
if self.currentLength > self.allowedLength:
    for i, length in enumerate(self.lengths):
        self.currentLength -= length
        self.lengths.pop(i)
        self.points.pop(i)
        if self.currentLength < self.allowedLength:
            break

# Check if snake ate the Food
rx, ry = self.foodPoint
if rx - self.wFood // 2 < cx < rx + self.wFood // 2 and \
    ry - self.hFood // 2 < cy < ry + self.hFood // 2:
    self.randomFoodLocation()
    self.allowedLength += 50
    self.score += 1
    print(self.score)

# Draw Snake
if self.points:
    for i, point in enumerate(self.points):
        if i != 0:
            cv2.line(imgMain, self.points[i - 1], self.points[i], (0, 0, 255), 20)
            cv2.circle(imgMain, self.points[-1], 20, (0, 255, 0), cv2.FILLED)

# Draw Food
imgMain = cvzone.overlayPNG(imgMain, self.imgFood,
                             (rx - self.wFood // 2, ry - self.hFood // 2))

cvzone.putTextRect(imgMain, f'Score: {self.score}', [50, 80],
                   scale=3, thickness=3, offset=10)

# Check for Collision
pts = np.array(self.points[:-2], np.int32)
pts = pts.reshape((-1, 1, 2))
cv2.polylines(imgMain, [pts], False, (0, 255, 0), 3)
minDist = cv2.pointPolygonTest(pts, (cx, cy), True)

if -0.2 <= minDist <= 0.2:
    print("Hit")
    self.gameOver = True
    self.points = [] # all points of the snake
    self.lengths = [] # distance between each point
    self.currentLength = 0 # total length of the snake
    self.allowedLength = 150 # total allowed Length

```

```

        self.previousHead = 0, 0 # previous head point
        self.randomFoodLocation()

    return imgMain

game = SnakeGameClass("Resources/Donut.png")

while True:
    success, img = cap.read()
    img = cv2.flip(img, 1)
    hands, img = detector.findHands(img, flipType=False)

    if hands:
        lmList = hands[0]['lmList']
        pointIndex = lmList[8][0:2]
        img = game.update(img, pointIndex)
    cv2.imshow("Image", img)
    key = cv2.waitKey(1)
    if key == ord('r'):
        game.gameOver = False

```

COMPARISION

(Mediapipe vs Basic ML Models)

Hand gesture recognition is an important task in computer vision that has various applications, including sign language recognition, virtual reality, and human-computer interaction. There are different approaches to recognize hand gestures, including using machine learning models and frameworks such as MediaPipe.

MediaPipe is an open-source framework that provides real-time solutions for various computer vision tasks, including hand gesture recognition. It uses a combination of computer vision techniques such as landmark detection, hand tracking, and classification to recognize hand gestures. The framework is easy to use, and it has a pre-trained model that can recognize a variety of hand gestures accurately.

On the other hand, machine learning models can also be used to recognize hand gestures. These models use various algorithms such as support vector machines, neural networks, and decision trees to learn the patterns and features of hand gestures. The models are trained on a dataset of hand gestures, and the accuracy of the model depends on the quality and diversity of the dataset.

Comparing MediaPipe and machine learning models for hand gesture recognition, both have their advantages and limitations.

MediaPipe is a good choice for real-time hand gesture recognition as it can process frames from a camera feed in real-time. This makes it ideal for applications that require immediate response, such as virtual reality and gaming. Additionally, the pre-trained model provided by MediaPipe is accurate and can recognize a variety of hand gestures.

Machine learning models, on the other hand, have the advantage of being more flexible and adaptable to specific use cases. They can be trained on custom datasets to recognize specific hand gestures, making them ideal for applications that require a high level of customization. Moreover, machine learning models can achieve higher accuracy than MediaPipe if they are trained on high-quality and diverse datasets.

In conclusion, both MediaPipe and machine learning models have their strengths and weaknesses, and the choice between them depends on the specific requirements of the application. MediaPipe is a good choice for real-time hand gesture recognition, while machine learning models are better for applications that require a high level of customization and accuracy.

LIMITATIONS

Hand gesture recognition using MediaPipe has several limitations that researchers and developers should be aware of. Some of these limitations are:

- 1- Lighting conditions:** Hand gesture recognition using MediaPipe can be affected by changes in lighting conditions. Poor lighting can result in low contrast and make it difficult for the system to detect and recognize hand gestures accurately.
- 2- Occlusions:** Hand gesture recognition using MediaPipe can be challenging when the hands are partially or fully occluded. For example, if one hand is partially covering the other, the system may not be able to recognize the gestures accurately.
- 3- Camera angle and position:** The position and angle of the camera can affect the accuracy of hand gesture recognition using MediaPipe. If the camera is not positioned correctly or is at an awkward angle, the system may not be able to recognize hand gestures accurately.
- 4- Limited gesture recognition:** Although the pre-trained model provided by MediaPipe can recognize several hand gestures accurately, it has its limitations. The system may not be able to recognize more complex or customized hand gestures, which can limit its usability in certain applications.
- 5- Processing power:** Hand gesture recognition using MediaPipe can be computationally intensive and requires a significant amount of processing power. This can limit its use in resource-constrained environments such as mobile devices.
- 6- Lack of diversity in the dataset:** The accuracy of hand gesture recognition using MediaPipe depends on the quality and diversity of the dataset used for training. If the dataset used for training is limited in diversity, the system may not be able to recognize hand gestures accurately in real-world scenarios.

In conclusion, hand gesture recognition using MediaPipe has several limitations that developers and researchers should consider. These limitations include lighting conditions, occlusions, camera angle and position, limited gesture recognition, processing power, and lack of diversity in the dataset. Despite these limitations, MediaPipe remains a powerful tool for real-time hand gesture recognition and has many potential applications in various fields.

FUTURE SCOPE

Hand gesture recognition using MediaPipe is a rapidly growing area of computer vision research, with many potential future applications. Here are some of the possible future scopes of hand gesture recognition using MediaPipe:

- 1- Improving accuracy:** Although the pre-trained model provided by MediaPipe is accurate, there is still scope for improvement. Future research can focus on developing more accurate hand gesture recognition models using MediaPipe by exploring new techniques, such as combining multiple models, using ensemble learning, and incorporating deep learning methods.
- 2- Real-time gesture recognition in complex environments:** MediaPipe currently works well in controlled environments, but there is scope for improvement in real-time gesture recognition in complex environments such as outdoors, crowded areas, or with complex backgrounds. Future research can focus on developing more robust hand tracking and gesture recognition algorithms that can handle complex and dynamic environments.
- 3- Gesture-based interaction with machines:** Hand gesture recognition using MediaPipe can be used for creating natural and intuitive human-machine interfaces. Future research can focus on developing more sophisticated gesture recognition algorithms that can interpret complex gestures and enable seamless interaction with machines, such as in robotics, automotive, and healthcare.
- 4- Sign language recognition:** Hand gesture recognition using MediaPipe can be used for sign language recognition. Future research can focus on developing models that can recognize more complex sign languages, incorporate facial expressions, and allow for natural conversation between deaf and hearing individuals.
- 5- Gesture-based gaming and virtual reality:** Hand gesture recognition using MediaPipe can be used to create immersive gaming and virtual reality experiences. Future research can focus on developing more advanced gesture recognition algorithms that can interpret complex hand gestures and allow for more natural and intuitive interaction with virtual environments.

CONCLUSION

In conclusion, hand gesture recognition using MediaPipe is a promising technology that has many potential applications in various fields such as human-computer interaction, robotics, gaming, and virtual reality. MediaPipe provides a simple and efficient way to detect and recognize hand gestures in real-time, making it an excellent choice for applications that require immediate response. The pre-trained models provided by MediaPipe are accurate and can recognize a variety of hand gestures, which makes it easy to use for developers and researchers.

However, the technology is still in its early stages, and there is a lot of room for improvement. Future research can focus on improving the accuracy of the models, handling complex environments, developing sophisticated human-machine interfaces, recognizing more complex sign languages, and enhancing gaming and virtual reality experiences.

Despite its limitations, hand gesture recognition using MediaPipe has the potential to transform the way we interact with machines and make it more intuitive and natural. With its ease of use and flexibility, MediaPipe is a powerful tool that can benefit researchers, developers, and end-users alike. Overall, hand gesture recognition using MediaPipe is an exciting area of research, and we can expect to see many new and innovative applications of this technology in the future.

BIBLIOGRAPHY

[1]- A parameter-less algorithm for tensor co-clustering.

Elena Battaglia & Ruggero G. Pensa

Volume 112

Pages 385–427

<https://link.springer.com/article/10.1007/s10994-021-06002-w>

[2]- Model-free inverse reinforcement learning with multi-intention, unlabeled, and overlapping demonstrations.

Ariyan Bighashdel, Pavol Jancura & Gijs Dubbelman

Volume 110

Pages 425-430

<https://link.springer.com/article/10.1007/s10994-022-06273-x>

[3]- Constrained regret minimization for multi-criterion multi-armed bandits.

Anmol Kagrecha, Jayakrishnan Nair & Krishna Jagannathan

Volume 112

Pages 431-458

<https://link.springer.com/article/10.1007/s10994-022-06291-9>

[4]- WINdowed TENsor decomposition for Densification Event Detection in time-evolving networks

Sofia Fernandes, Hadi Fanaee-T, João Gama, Leo Tišljarić & Tomislav Šmuc

Volume 112

Pages 459-481

<https://link.springer.com/article/10.1007/s10994-021-05979-8>

[5]- Learning multi-agent coordination through connectivity-driven communication.

Emanuele Pesce & Giovanni Montana

Volume 112

Pages 483-514

<https://link.springer.com/article/10.1007/s10994-022-06286-6>

[6]- Feed-Forward Neural-Symbolic Learner.

Daniel Cunningham, Mark Law, Jorge Lobo & Alessandra Russo

Volume 112

Pages 515-569

<https://link.springer.com/article/10.1007/s10994-022-06278-6>

[7]- Efficient learning of large sets of locally optimal classification rules.

Van Quoc Phuong Huynh, Johannes Fürnkranz & Florian Beck

Volume 112

Pages 571-610

<https://link.springer.com/article/10.1007/s10994-022-06290-w>

[8]- Circular-symmetric correlation layer.

Bahar Azari & Deniz Erdoğan

Volume 112

Pages 611-631

<https://link.springer.com/article/10.1007/s10994-022-06288-4>

[9]- Robust federated learning under statistical heterogeneity via hessian-weighted aggregation.

Adnan Ahmad, Wei Luo & Antonio Robles-Kelly

Volume 112

Pages 633-654

<https://link.springer.com/article/10.1007/s10994-022-06292-8>

[10]- Hierarchically structured task-agnostic continual learning.

Heinke Hihn & Daniel A. Braun

Volume 112

Pages 655-686

<https://link.springer.com/article/10.1007/s10994-022-06283-9>

[11]- Relational data embeddings for feature enrichment with background information.

Alexis Cvetkov-Iliev, Alexandre Allauzen & Gaël Varoquaux

Volume 112

Pages 687-720

<https://link.springer.com/article/10.1007/s10994-022-06277-7>

[12]- Embedding to reference t-SNE space addresses batch effects in single-cell classification.

Pavlin G. Poličar, Martin Stražar & Blaž Zupan

Volume 112

Pages 721-740

<https://link.springer.com/article/10.1007/s10994-021-06043-1>

[13]- Time and space complexity of deterministic and nondeterministic decision trees.

Mikhail Moshkov

Volume 91

Pages 45-74

<https://link.springer.com/article/10.1007/s10472-022-09814-1>

[14]- Neural networks in Fréchet spaces.

Fred Espen Benth, Nils Detering & Luca Galimberti

Volume 91

Pages 75-103

<https://link.springer.com/article/10.1007/s10472-022-09824-z>

[15]- Knowledge forgetting in propositional μ -calculus.

Renyan Feng, Yisong Wang, Ren Qian, Lei Yang & Panfeng Chen

Volume 91

Pages 1-43

<https://link.springer.com/article/10.1007/s10472-022-09803-4>