Ministry of Science and Higher Education of the Republic of Kazakhstan
L.N. Gumilyov Eurasian National University

Faculty of Information Technology
Department of Information Systems

MIDTERM 2: SMART CURSOR WITH AI

Done by: Akbayan, Aruzhan, Inkara

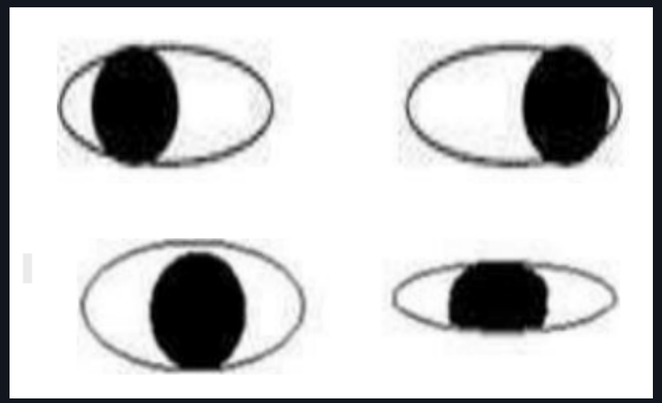
INTRODUCTION

PROJECT OBJECTIVE:

To develop a system that allows mouse cursor control using hand gestures tracked through a webcam, without the use of additional controllers.

👁️ Cursor control using your eyes

This is a technology that allows you to control mouse movement by tracking the direction of the user's gaze. A camera captures the position of the pupils and the software interprets this as coordinates on the screen. This method provides hands-free interaction and can be useful for people with disabilities or in systems without physical input.

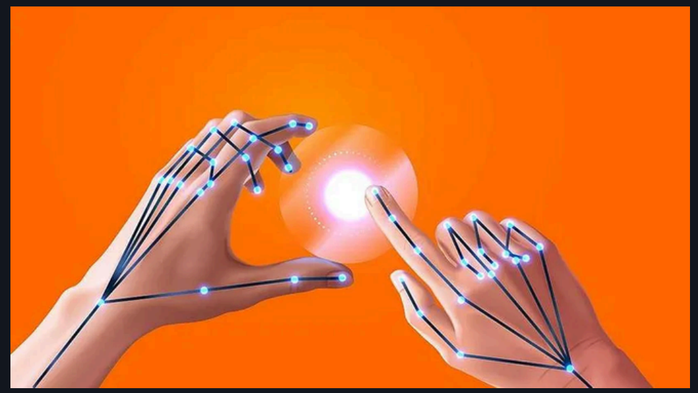


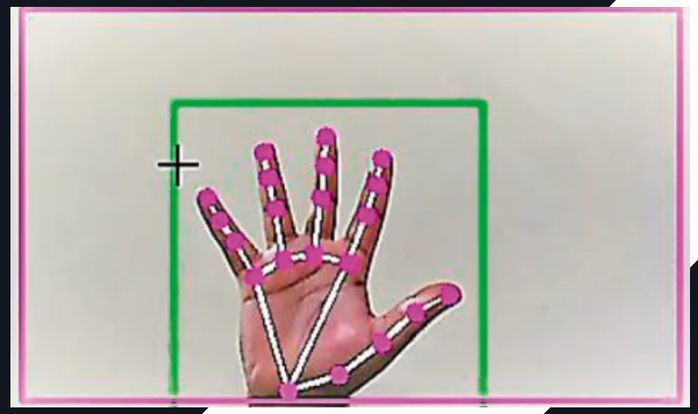
✋ Cursor control using hand

This method is based on recognizing the position and movement of fingers through a normal camera. For example, the index finger is used to move the cursor, and touching it with the thumb is used to simulate a mouse click. This is a non-contact, intuitive way of control that does not require any additional devices.

WHY DO WE USE THE VIRTUAL MOUSE?

The proposed AI virtual mouse system can be used to overcome problems in the real world such as situations where there is no space to use a physical mouse and also for persons who have problems in their hands and are not able to control a physical mouse. Also, amidst the COVID-19 situation, it is not safe to use the devices by touching them because it may result in a possible situation of the spread of the virus by touching the devices, so the proposed AI virtual mouse can be used to overcome these problems since hand gestures and hand Tip detection is used to control the PC mouse functions by using a webcam or a built-in camera.





TECHNOLOGIES USED:

🐍 Python
The programming language in which the entire project is written.
Easy to use and ideal for working with video, AI and computer vision.

✋ MediaPipe (from Google)
A library for gesture recognition. The project:
- Detects finger position and movement;
- tracks the hand in real time;
- helps to understand which fingers are raised and whether they are touching each other.

🔢 NumPy
A library for scientific computing. The project:
helps “smooth out” cursor movements;
- translates coordinates from camera to screen coordinates.

📷 OpenCV
Computer Vision Library. Used for:
- camera video capture;
- image conversion;
- display graphics (frames, circles, text on video).

🖱️ AutoPy
A library for mouse and keyboard control on your computer.
The project:
- moves the mouse cursor across the screen;
- simulates mouse click when fingers touch.

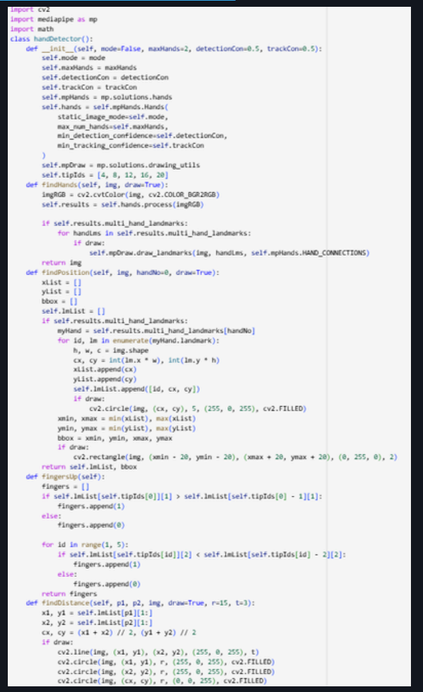
📐 math
A standard Python module. Used for:
calculating the distance between fingers;
determine if the fingers are close enough to click.

📦 PROJECT STRUCTURE:

1. main.py - main executable code
   - Camera startup, video display.
   - Cursor control using eyes and hands.
   - Uses HandDetector and EyeTracker classes.
   - Defines gestures for mouse movements and clicks.
   - Calls all basic methods: hand capture, eye capture, mouse control.

2. hands_module.py - module for hand detection
   1. HandDetector class
   2. Methods:
   - findHands() - hand detection and skeleton rendering.
   - findPosition() - coordinates of key points.
   - fingersUp() - determines which fingers are raised.
   - findDistance(p1, p2) - distance between fingers.
   - isThumbAndIndexTouching() - determines whether the thumb and index finger are touching.

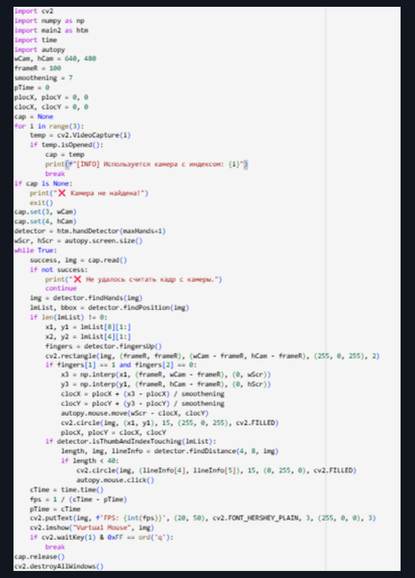
3. eye_module.py - module for eye tracking
   1. EyeTracker class
   2. Uses the MediaPipe Face Mesh to determine the position of the eyes.
   3. Methods:
   - get_eye_position() - returns the coordinates of the center of the eye.
   - Handles head/eye position for cursor control.



```python
import cv2
import mediapipe as mp
import math
class handDetector():
    def __init__(self, mode=False, maxHands=2, detectionCon=0.5, trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.trackCon = trackCon
        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(
            static_image_mode=self.mode,
            max_num_hands=self.maxHands,
            min_detection_confidence=self.detectionCon,
            min_tracking_confidence=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)

        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_CONNECTIONS)
        return img
    def findPosition(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):
                h, w, c = img.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                xList.append(cx)
                yList.append(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, (xmin - 20, ymin - 20), (xmax + 20, ymax + 20), (0, 255, 0), 2)
        return self.lmList, bbox
    def fingersUp(self):
        fingers = []
        if self.lmList[self.tipIds[0]][1] > self.lmList[self.tipIds[0] - 1][1]:
            fingers.append(1)
        else:
            fingers.append(0)

        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, r=15, t=3):
        x1, y1 = self.lmList[p1][1:]
        x2, y2 = self.lmList[p2][1:]
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
        if draw:
            cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), t)
            cv2.circle(img, (x1, y1), r, (255, 0, 255), cv2.FILLED)
            cv2.circle(img, (x2, y2), r, (255, 0, 255), cv2.FILLED)
            cv2.circle(img, (cx, cy), r, (0, 0, 255), cv2.FILLED)
```



```python
import cv2
import numpy as np
import main2 as htm
import time
import autopy
wCam, hCam = 640, 480
frameR = 100
smoothening = 7
pTime = 0
plocX, plocY = 0, 0
clocX, clocY = 0, 0
cap = None
for i in range(3):
    temp = cv2.VideoCapture(i)
    if temp.isOpened():
        cap = temp
        print(f"[INFO] Используется камера с индексом: {i}")
        break
if cap is None:
    print("❌ Камера не найдена!")
    exit()
cap.set(3, wCam)
cap.set(4, hCam)
detector = htm.handDetector(maxHands=1)
wScr, hScr = autopy.screen.size()
while True:
    success, img = cap.read()
    if not success:
        print("❌ Не удалось считать кадр с камеры.")
        continue
    img = detector.findHands(img)
    lmList, bbox = detector.findPosition(img)
    if len(lmList) != 0:
        x1, y1 = lmList[8][1:]
        x2, y2 = lmList[4][1:]
        fingers = detector.fingersUp()
        cv2.rectangle(img, (frameR, frameR), (wCam - frameR, hCam - frameR), (255, 0, 255), 2)
        if fingers[1] == 1 and fingers[2] == 0:
            x3 = np.interp(x1, (frameR, wCam - frameR), (0, wScr))
            y3 = np.interp(y1, (frameR, hCam - frameR), (0, hScr))
            clocX = plocX + (x3 - plocX) / smoothening
            clocY = plocY + (y3 - plocY) / smoothening
            autopy.mouse.move(wScr - clocX, clocY)
            cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
            plocX, plocY = clocX, clocY
        if detector.isThumbAndIndexTouching(lmList):
            length, img, lineInfo = detector.findDistance(4, 8, img)
            if length < 40:
                cv2.circle(img, (lineInfo[4], lineInfo[5]), 15, (0, 255, 0), cv2.FILLED)
                autopy.mouse.click()
    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    cv2.putText(img, f'FPS: {int(fps)}', (20, 50), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 3)
    cv2.imshow("Virtual Mouse", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

```python
import cv2
import numpy as np
import mediapipe as mp
import time
import autopy
wCam, hCam = 640, 480
smoothening = 3
blink_threshold = 30
pTime = 0
plocX, plocY = 0, 0
clocX, clocY = 0, 0
wScr, hScr = autopy.screen.size()
mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh(refine_landmarks=True)
mp_drawing = mp.solutions.drawing_utils
cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
while True:
    success, img = cap.read()
    if not success:
        continue
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = face_mesh.process(imgRGB)
    if results.multi_face_landmarks:
        face_landmarks = results.multi_face_landmarks[0]
        ih, iw, _ = img.shape
        left_eye = [33, 159]
        right_eye = [362, 386]
        iris = 468
        ix, iy = int(face_landmarks.landmark[iris].x * iw), int(face_landmarks.landmark[iris].y * ih)
        x3 = np.interp(ix, (0, wCam), (0, wScr))
        y3 = np.interp(iy, (0, hCam), (0, hScr))
        clocX = plocX + (x3 - plocX) / smoothening
        clocY = plocY + (y3 - plocY) / smoothening
        clocX = np.clip(clocX, 0, wScr)
        clocY = np.clip(clocY, 0, hScr)
        autopy.mouse.move(wScr - clocX, clocY)
        plocX, plocY = clocX, clocY
        def get_eye_distance(p1, p2):
            x1 = int(face_landmarks.landmark[p1].x * iw)
            y1 = int(face_landmarks.landmark[p1].y * ih)
            x2 = int(face_landmarks.landmark[p2].x * iw)
            y2 = int(face_landmarks.landmark[p2].y * ih)
            return np.linalg.norm([x2 - x1, y2 - y1])
        left_blink = get_eye_distance(*left_eye)
        right_blink = get_eye_distance(*right_eye)
        if left_blink < blink_threshold and right_blink < blink_threshold:
            autopy.mouse.click()
            time.sleep(0.2)
    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    cv2.putText(img, f'FPS: {int(fps)}', (20, 50),
                cv2.FONT_HERSHEY_PLAIN, 2, (0, 255, 0), 2)
    cv2.imshow("AI Eye Mouse", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

📏 CONTROL LOGIC:

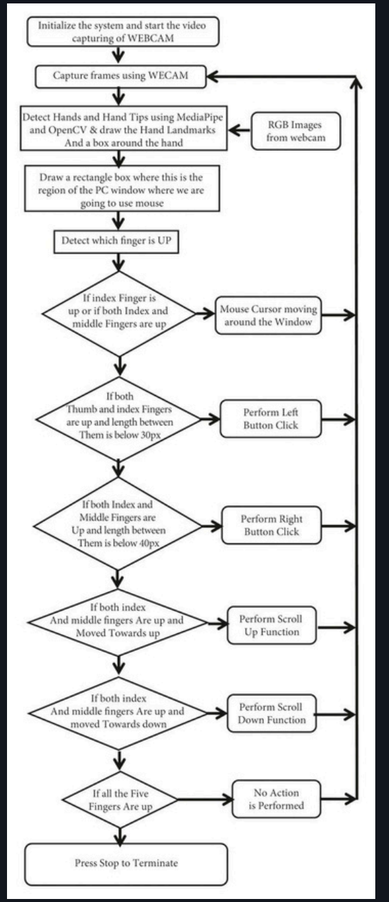
- Mouse move – only the index finger is raised.
- Mouse click – thumb and index finger touch.

🖼️ VISUALIZATION

- Displays the skeleton of the arm using MediaPipe.
- A rectangle (frameR) limits the active control area of the hand.
- The position of the fingers is visualized by circles.
- The FPS (frames per second) in the upper left corner is displayed.
- When thumb and index finger are touched, a green circle is drawn at the point of touch (click indication).

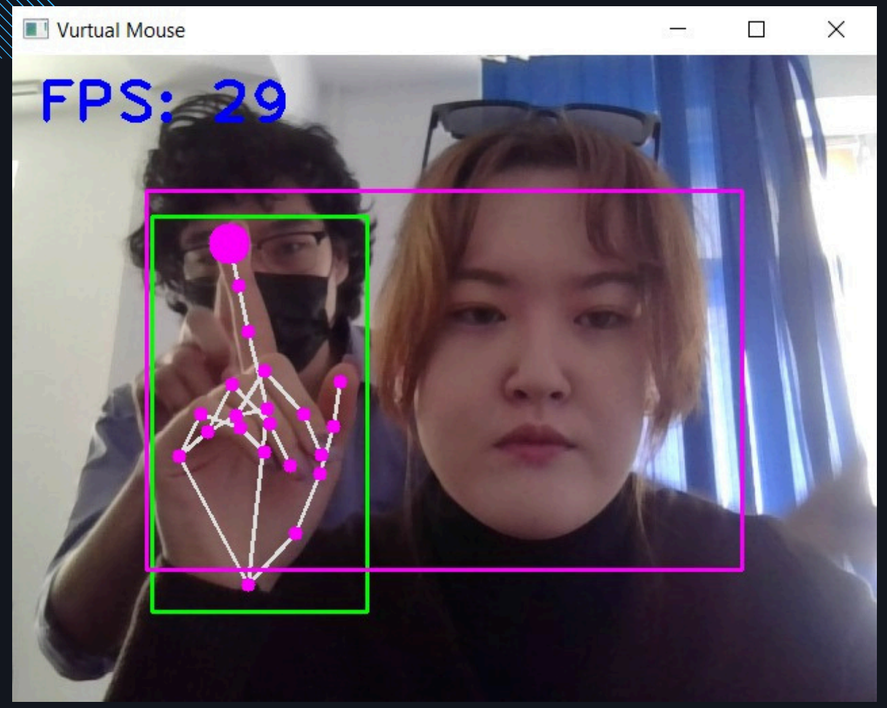
TEST RESULTS:

- Gesture recognition accuracy: high, especially in good lighting conditions. The MediaPipe module successfully recognizes hands and fingers with accuracy, which allows accurate cursor control.
- FPS: 20 to 30 frames per second. This is optimal for smooth cursor interaction, with no noticeable lag.
- Latency: minimal. Smooth cursor movement and clicks are provided using anti-aliasing and interpolation, making interaction more natural.



FLOW CHART

THE RESULT



Then continuing in real time. Enjoy the show

NUMBER PLATE RECOGNATION

Authors: Kalybek Aruzhan, Zhumabek Akbayan, Suleimenova Inkara
Supervisor: Prof. Dr. Tamara Kokenovna

1. Background

To develop a system that allows mouse cursor control using hand gestures tracked through a webcam, without the use of additional controllers.

Cursor control using your eyes
This is a technology that allows you to control mouse movement by tracking the direction of the user's gaze. A camera captures the position of the pupils and the software interprets this as coordinates on the screen. This method provides hands-free interaction and can be useful for people with disabilities or in systems without physical input.

Cursor control using hand gestures
This method is based on recognizing the position and movement of fingers through a normal camera.

2. Objectives

Aim
Our project focuses on developing an AI-powered Virtual Mouse system using computer vision. The goal is to create a smart, touchless interface that allows users to:

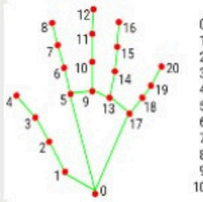
- Control the computer cursor using hand gestures
- Simulate mouse clicks and actions through gesture recognition
- Navigate and interact more intuitively using eye tracking for direction and focus detection

Research Question
How accurately can AI-based computer vision models detect and interpret hand gestures and eye movements, and how effectively can these inputs replace traditional mouse control in real-time applications?
Objective
To build an automated, user-friendly, and hands-free virtual mouse system that enhances accessibility, efficiency, and user experience. By integrating gesture and eye tracking, our project aims to contribute to next-generation human-computer interaction, useful for general use, accessibility solutions, and smart environments.

3. Methods

Our project focuses on developing an AI Virtual Mouse using computer vision and deep learning techniques. The system allows users to control the mouse cursor using hand gestures detected via a webcam, providing a touchless and intuitive human-computer interaction experience.
The core of the system relies on mediapipe for real-time hand landmark detection, which tracks finger movements and gestures with high accuracy. Specific gestures, such as raising the index finger, are used to move the cursor, while pinch gestures simulate left and right mouse clicks. The cursor position is continuously updated based on the fingertip coordinates mapped to the screen dimensions.
To enhance user interaction and accessibility, we extended the functionality by adding eye tracking, allowing the system to detect the user's gaze direction. This addition supports more precise control and can serve as an alternative or complement to hand tracking, particularly useful in accessibility scenarios.
The project demonstrates how AI and computer vision can replace traditional input devices, offering a more natural, hygienic, and modern way to interact with computers. With further improvements, this system can be used in smart environments, assistive technologies, and gesture-based control interfaces.

3a. Model



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

MediaPipe Hands is a pre-trained model that detects and tracks hands in real time using a regular webcam. It identifies 21 hand landmarks, including fingertips, joints, and the palm base. This allows precise gesture recognition, which can be used for applications like virtual mouse control or sign language interpretation.

3b. Analysis

The MediaPipe Hands solution by Google provides a pre-trained model for real-time hand detection and tracking, making it ideal for projects like the AI Virtual Mouse. It detects and tracks 21 hand landmarks, including fingertips and joints, using just a standard webcam. This enables accurate gesture recognition without the need for custom datasets or model training.
MediaPipe Hands works across different lighting conditions and backgrounds, offering reliable performance for interactive applications. Its lightweight and cross-platform nature makes it especially suitable for building touchless interfaces, gesture-controlled systems, and accessibility tools like the AI Virtual Mouse, which allows users to control a computer using natural hand movements.

4. Test Result +

- Gesture recognition accuracy: high, especially in good lighting conditions. The MediaPipe module successfully recognizes hands and fingers with accuracy, which allows accurate cursor control.
- Latency: minimal. Smooth cursor movement and clicks are provided using anti-aliasing and interpolation, making interaction more natural.
- FPS: 20 to 30 frames per second. This is optimal for smooth cursor interaction, with no noticeable lag.
- No training required: Since MediaPipe Hands is a pre-trained model, it does not require additional training or labeled datasets, which significantly speeds up development time for applications like the AI Virtual Mouse.
- Lightweight and efficient: The module is optimized for performance, making it suitable even for systems with limited hardware resources, such as laptops or embedded devices.

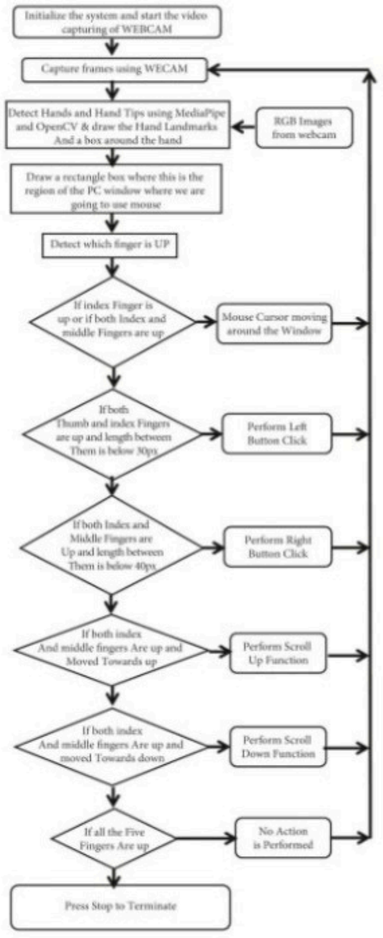
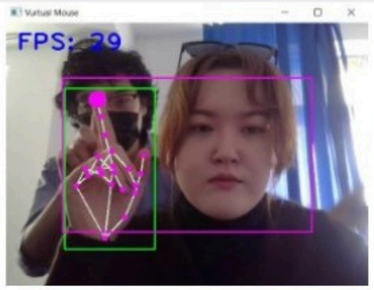


Fig 1. Flow chart



5. Conclusion

The Smart Cursor with AI project demonstrates the potential of combining hand and eye tracking technologies to create a fully touchless and intuitive human-computer interaction system. By leveraging OpenCV, MediaPipe, and Autopy, we achieved accurate gesture and gaze detection, enabling smooth and responsive cursor control. This innovative approach not only enhances user experience but also offers a practical solution for individuals with limited mobility or motor impairments. Our results confirm that AI-driven interaction can be both efficient, accessible, and ready for real-world applications, paving the way for smarter, more inclusive digital environments.

Furthermore, the integration of hand and eye tracking with AI offers significant advantages in terms of accessibility and convenience. This technology can be easily adapted to various environments, including healthcare, education, and home automation, providing users with more intuitive and hands-free interaction options. As the system continues to evolve, it has the potential to redefine how we interact with devices, reducing reliance on traditional input methods and opening up new possibilities for a wide range of applications. With further optimization, this system could become an essential tool in creating more inclusive and user-friendly digital interfaces.

References

Article 1
Article 2
Article 3

our contact:
akbayan.zhumabek@mail.ru
s.inkara.a04@gmail.com
aruzhan.kalybek26@gmail.com

CONCLUSION

In the Smart Cursor with AI project, we implemented cursor control using hands and eyes. The system is based on OpenCV, MediaPipe and Autopy. It accurately detects gestures and eye movements, allowing you to control the mouse without touching it. This approach is convenient and can be useful for people with disabilities. The project has shown good results and can be extended in the future.

THANK YOU

...

OUR CONTACT:
AKBAYAN.ZHUMABEK@MAIL.RU
S.INKARA.A04@GMAIL.COM
ARUZHAN.KALYBEK26@GMAIL.COM