实战项目 1：AI 手势虚拟拖拽方块

步骤：

1. OpenCV 获取视频流
2. 在画面上画一个方块
3. 通过 mediapipe 获取手指关键点坐标

   （ https://google.github.io/mediapipe/solutions/hands ）
4. 判断手指是否在方块上
5. 根据食指和中指指尖的坐标，利用勾股定理计算距离，当距离较小且都落在矩形内，则触发拖拽（矩形变色）；
6. 矩形跟着手指动；
7. 两指放开，则矩形停止移动

完善：画面显示 FPS 等信息

标准答案：

```
# 导入 OpenCV
import cv2
# 导入 mediapipe
import mediapipe as mp


# 导入其他依赖包
import time
import math


# 方块管理类
class SquareManager:
    def __init__(self, rect_width):


        # 方框长度
```

```
        self.rect_width = rect_width


        # 方块 list
        self.square_count = 0
        self.rect_left_x_list = []
        self.rect_left_y_list = []
        self.alpha_list = []


        # 中指与矩形左上角点的距离
        self.L1 = 0
        self.L2 = 0


        # 激活移动模式
        self.drag_active = False


        # 激活的方块 ID
        self.active_index = -1


    # 创建一个方块，但是没有显示
    def create(self, rect_left_x, rect_left_y, alpha=0.4):
        self.rect_left_x_list.append(rect_left_x)
        self.rect_left_y_list.append(rect_left_y)
        self.alpha_list.append(alpha)
        self.square_count += 1


    # 更新位置
    def display(self, class_obj):
        for i in range(0, self.square_count):
```

```
            x = self.rect_left_x_list[i]
            y = self.rect_left_y_list[i]
            alpha = self.alpha_list[i]

            overlay = class_obj.image.copy()

            if (i == self.active_index):
                cv2.rectangle(overlay, (x, y), (x + self.rect_width, y + self.rect_width),
(255, 0, 255), -1)
            else:
                cv2.rectangle(overlay, (x, y), (x + self.rect_width, y + self.rect_width),
(255, 0, 0), -1)

            # Following line overlays transparent rectangle over the self.image
            class_obj.image = cv2.addWeighted(overlay, alpha, class_obj.image, 1 - alpha,
0)

    # 判断落在哪个方块上，返回方块的 ID
    def checkOverlay(self, check_x, check_y):
        for i in range(0, self.square_count):
            x = self.rect_left_x_list[i]
            y = self.rect_left_y_list[i]

            if (x < check_x < (x + self.rect_width)) and (y < check_y < (y +
self.rect_width)):
                # 保存被激活的方块 ID
                self.active_index = i
```

```
                return i


        return -1


    # 计算与指尖的距离
    def setLen(self, check_x, check_y):
        # 计算距离
        self.L1 = check_x - self.rect_left_x_list[self.active_index]
        self.L2 = check_y - self.rect_left_y_list[self.active_index]


    # 更新方块
    def updateSquare(self, new_x, new_y):
        # print(self.rect_left_x_list[self.active_index])
        self.rect_left_x_list[self.active_index] = new_x - self.L1
        self.rect_left_y_list[self.active_index] = new_y - self.L2


# 识别控制类
class HandControlVolume:
    def __init__(self):
        # 初始化 medialpipe
        self.mp_drawing = mp.solutions.drawing_utils
        self.mp_drawing_styles = mp.solutions.drawing_styles
        self.mp_hands = mp.solutions.hands


        # 中指与矩形左上角点的距离
        self.L1 = 0
        self.L2 = 0
```

```
        # image 实例，以便另一个类调用
        self.image = None


    # 主函数
    def recognize(self):
        # 计算刷新率
        fpsTime = time.time()


        # OpenCV 读取视频流
        cap = cv2.VideoCapture(0)
        # 视频分辨率
        resize_w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        resize_h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))


        # 画面显示初始化参数
        rect_percent_text = 0


        # 初始化方块管理器
        squareManager = SquareManager(150)


        # 创建多个方块
        for i in range(0, 5):
            squareManager.create(200 * i + 20, 200, 0.6)


        with self.mp_hands.Hands(min_detection_confidence=0.7,
                                 min_tracking_confidence=0.5,
                                 max_num_hands=2) as hands:
            while cap.isOpened():
```

```
# 初始化矩形
success, self.image = cap.read()
self.image = cv2.resize(self.image, (resize_w, resize_h))

if not success:
    print("空帧.")
    continue

# 提高性能
self.image.flags.writeable = False
# 转为 RGB
self.image = cv2.cvtColor(self.image, cv2.COLOR_BGR2RGB)
# 镜像
self.image = cv2.flip(self.image, 1)
# mediapipe 模型处理
results = hands.process(self.image)

self.image.flags.writeable = True
self.image = cv2.cvtColor(self.image, cv2.COLOR_RGB2BGR)
# 判断是否有手掌
if results.multi_hand_landmarks:
    # 遍历每个手掌
    for hand_landmarks in results.multi_hand_landmarks:
        # 在画面标注手指
        self.mp_drawing.draw_landmarks(
            self.image,
            hand_landmarks,
```

```
                            self.mp_hands.HAND_CONNECTIONS,
                            self.mp_drawing_styles.get_default_hand_landmarks_style(),
                            self.mp_drawing_styles.get_default_hand_connections_style())

                        # 解析手指，存入各个手指坐标
                        landmark_list = []

                        # 用来存储手掌范围的矩形坐标
                        paw_x_list = []
                        paw_y_list = []
                        for landmark_id, finger_axis in enumerate(
                                hand_landmarks.landmark):
                            landmark_list.append([
                                landmark_id, finger_axis.x, finger_axis.y,
                                finger_axis.z
                            ])
                            paw_x_list.append(finger_axis.x)
                            paw_y_list.append(finger_axis.y)
                        if landmark_list:
                            # 比例缩放到像素
                            ratio_x_to_pixel = lambda x: math.ceil(x * resize_w)
                            ratio_y_to_pixel = lambda y: math.ceil(y * resize_h)

                            # 设计手掌左上角、右下角坐标
                            paw_left_top_x, paw_right_bottom_x = map(ratio_x_to_pixel,
                                                                     [min(paw_x_list),
max(paw_x_list)])
                            paw_left_top_y, paw_right_bottom_y = map(ratio_y_to_pixel,
```

```
                                                                    [min(paw_y_list),
max(paw_y_list)])

                            # 给手掌画框框
                            cv2.rectangle(self.image, (paw_left_top_x - 30, paw_left_top_y
- 30),
                                          (paw_right_bottom_x + 30, paw_right_bottom_y +
30), (0, 255, 0), 2)

                            # 获取中指指尖坐标
                            middle_finger_tip = landmark_list[12]
                            middle_finger_tip_x = ratio_x_to_pixel(middle_finger_tip[1])
                            middle_finger_tip_y = ratio_y_to_pixel(middle_finger_tip[2])

                            # 获取食指指尖坐标
                            index_finger_tip = landmark_list[8]
                            index_finger_tip_x = ratio_x_to_pixel(index_finger_tip[1])
                            index_finger_tip_y = ratio_y_to_pixel(index_finger_tip[2])
                            # 中间点
                            between_finger_tip = (middle_finger_tip_x + index_finger_tip_x)
// 2, (
                                                  middle_finger_tip_y + index_finger_tip_y) // 2
                            # print(middle_finger_tip_x)
                            thumb_finger_point = (middle_finger_tip_x, middle_finger_tip_y)
                            index_finger_point = (index_finger_tip_x, index_finger_tip_y)
                            # 画指尖 2 点
                            circle_func = lambda point: cv2.circle(self.image, point, 10,
(255, 0, 255), -1)
```

```
                            self.image = circle_func(thumb_finger_point)
                            self.image = circle_func(index_finger_point)
                            self.image = circle_func(between_finger_tip)
                            # 画 2 点连线
                            self.image = cv2.line(self.image, thumb_finger_point,
index_finger_point, (255, 0, 255), 5)
                            # 勾股定理计算长度
                            line_len = math.hypot((index_finger_tip_x -
middle_finger_tip_x),
                                                  (index_finger_tip_y -
middle_finger_tip_y))
                            # 将指尖距离映射到文字
                            rect_percent_text = math.ceil(line_len)

                            # 激活模式，需要让矩形跟随移动
                            if squareManager.drag_active:
                                # 更新方块
                                squareManager.updateSquare(between_finger_tip[0],
between_finger_tip[1])
                                if (line_len > 100):
                                    # 取消激活
                                    squareManager.drag_active = False
                                    squareManager.active_index = -1

                            elif (line_len < 100) and
(squareManager.checkOverlay(between_finger_tip[0],

between_finger_tip[1]) != -1) and (
```

```
                                    squareManager.drag_active == False):
                                # 激活
                                squareManager.drag_active = True
                                # 计算距离
                                squareManager.setLen(between_finger_tip[0],
between_finger_tip[1])

                # 显示方块，传入本实例，主要为了半透明的处理
                squareManager.display(self)

                # 显示距离
                cv2.putText(self.image, "Distance:" + str(rect_percent_text), (10, 120),
cv2.FONT_HERSHEY_PLAIN, 3,
                            (255, 0, 0), 3)

                # 显示当前激活
                cv2.putText(self.image, "Active:" + (
                    "None" if squareManager.active_index == -1 else
str(squareManager.active_index)), (10, 170),
                            cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 3)

                # 显示刷新率 FPS
                cTime = time.time()
                fps_text = 1 / (cTime - fpsTime)
                fpsTime = cTime
                cv2.putText(self.image, "FPS: " + str(int(fps_text)), (10, 70),
                            cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 3)
                # 显示画面
```

```
                # self.image = cv2.resize(self.image, (resize_w//2, resize_h//2))
                cv2.imshow('virtual drag and drop', self.image)

                if cv2.waitKey(5) & 0xFF == 27 :
                    break
            cap.release()

# 开始程序
control = HandControlVolume()
control.recognize()
```

自己答案：

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

video = cv2.VideoCapture(0)

#激活方块标志
sign = False

#获取视频长宽
video_width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
video_height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))

#正方形相关参数
rec_width = 150
rec_height = 150
```

```
dis_x = now_posx = start_posx = 50

dis_y = now_posy = start_posy = 50


#mediapipe 相关参数

mp_drawing = mp.solutions.drawing_utils

mp_drawing_styles = mp.solutions.drawing_styles

mp_hands = mp.solutions.hands


hands = mp_hands.Hands(

    model_complexity=0,            #模型复杂度

    min_detection_confidence=0.5, #检测最小置信度

    min_tracking_confidence=0.5)   #手部跟踪最小置信度


start_time = time.time()


def rectangle_draw(start, img, x, y):

    #情况一：画黄色矩形

    if start:

        img = cv2.rectangle(img, (x, y), (x + rec_width, y + rec_height), (0, 255, 0), -1)

    #情况二：画绿色矩形

    elif start == False:

        img = cv2.rectangle(img, (x, y), (x + rec_width, y + rec_height), (255, 255, 0), -
1)


while True:

    ret, frame = video.read()

    #将 cv2 的 BGR 格式转换成 RGB

    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

```
    # 镜像
    frame = cv2.flip(frame,1)


    # if sign == False:
    #     rec_x = dis_x
    #     rec_y = dis_y
    #     #绘制实心正方形
    #     frame = cv2.rectangle(frame, (rec_x, rec_y), (rec_x + rec_width, rec_y +
rec_height), (0, 255, 0), -1)
    results = hands.process(frame)


    if results.multi_hand_landmarks:
      for hand_landmarks in results.multi_hand_landmarks:
        #获得食指坐标 x，y
        finger1_x = int(hand_landmarks.landmark[8].x * video_width)
        finger1_y = int(hand_landmarks.landmark[8].y * video_height)
        #print(finger1_x, finger1_y)


        #获得中指坐标 x，y
        finger2_x = int(hand_landmarks.landmark[12].x * video_width)
        finger2_y = int(hand_landmarks.landmark[12].y * video_height)
        #print(finger2_x, finger2_y)


        #验证是否是食指、中指
        # frame = cv2.circle(frame, (finger1_x, finger1_y), 30, (255, 0, 0), -1)
        # frame = cv2.circle(frame, (finger2_x, finger2_y), 30, (255, 0, 0), -1)


        #获得中指和食指的距离—勾股定理
```

```
        # distance =math.sqrt( (finger2_x - finger1_x)**2 + (finger2_x - finger1_x)**2)
        distance = math.hypot((finger1_x - finger2_x), (finger1_y - finger2_y))
        #print(distance)

        #情况一：触发移动条件，方块跟随手指
        if distance < 45:
            if (finger1_x < now_posx + rec_width) and (finger1_x > now_posx) and
(finger1_y > now_posy) and (finger1_y < now_posy + rec_height):
                if sign == False:
                    sign = True
                    #计算矩形左上角点与食指的相对距离
                    dis_x = finger1_x - now_posx
                    dis_y = finger1_y - now_posy
        #情况二：解除/原始状态
        elif distance > 240:
            sign = False

        #避免方块移出视频画面
        if now_posx + rec_width > video_width: #右限位
            sign = False
            now_posx = video_width - rec_width
        elif now_posx < 0: #左限位
            sign = False
            now_posx = 0
        elif now_posy < 0: #上限位
            sign = False
            now_posy = 0
        elif now_posy + rec_height  > video_height: #下限位
```

```
            sign = False
            now_posy = video_height - rec_height

        #移动跟随算法
        if sign:
            now_posx = finger1_x - dis_x
            now_posy = finger1_y - dis_y

        #绘制手指关键点
        mp_drawing.draw_landmarks(
            frame,
            hand_landmarks,
            mp_hands.HAND_CONNECTIONS,
            mp_drawing_styles.get_default_hand_landmarks_style(),
            mp_drawing_styles.get_default_hand_connections_style())

    #矩形半透明处理
    overlay = frame.copy()
    rectangle_draw(sign, frame, now_posx, now_posy)
    frame = cv2.addWeighted(overlay, 0.5, frame, 1 - 0.5, 0)

    #左上角显示帧率
    now_time = time.time()
    fps = int(1/(now_time - start_time))
    start_time = now_time
    font = cv2.FONT_HERSHEY_PLAIN
    frame = cv2.putText(img = frame, text = str(fps), org = (20, 50), fontFace = font,
fontScale = 3, color = (0, 255, 255), thickness = 3, lineType = cv2.LINE_AA)
```

```
    #恢复颜色
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
    #显示视频画面，设置退出条件
    cv2.imshow("Mac Camera", frame)
    if cv2.waitKey(10) & 0xFF == 27:
        break

#释放句柄
video.release()
cv2.destroyAllWindows()
```

实现效果：