课程设计报告

课设题目：手势控制鼠标与键盘

完成人列表：

学生姓名	学号	分工及贡献	贡献率（%）
李政廉	2020010904022	结构设计和代码编写	33.33333
黄鑫杰	2020010902012	文档编写和算法原理	33.33333
傅英伦	2020010904014	ppt 制作和算法原理	33.33333

完成日期：2021.11.19

指导教师：庄杰老师

目录

1 系统简介

本项目通过利用 Mediapipe、opencv、autopy 等库，实现了利用电脑前置摄像头对手势以及移动方向进行识别，从而实现控制鼠标移动，单机鼠标左右键，以及键盘快捷键指令（比如上下翻页）。

2 需求分析

2.1 功能与性能

功能：调用 PC 前置摄像头，识别手势以及手指移动方向，从而实现对鼠标以及键盘的人机交互手势控制。

性能：便于操作，鼠标移动不会过于灵敏也不会过于迟钝；同时保证不会产生由于判断条件不够周全而导致的鼠标连点，规避掉一系列误操作。

2.2 软硬件平台要求

2.2.1 软件要求

本系统基于 Python 语言编写，使用 pycharm 编译器实现，通过安装 Mediapipe 库、opencv 库以及 autopy 等工具包。使用前需据上述条件搭建相关环境。

2.2.2 硬件需求

本系统在 x64 架构的计算机上运行稳定，同时至少需要性能中等的 cpu、gpu 以及内存支持。Cpu 和 gpu 的性能较差会导致显示的画面较为卡顿。

3 基本原理

3.1 所用模型

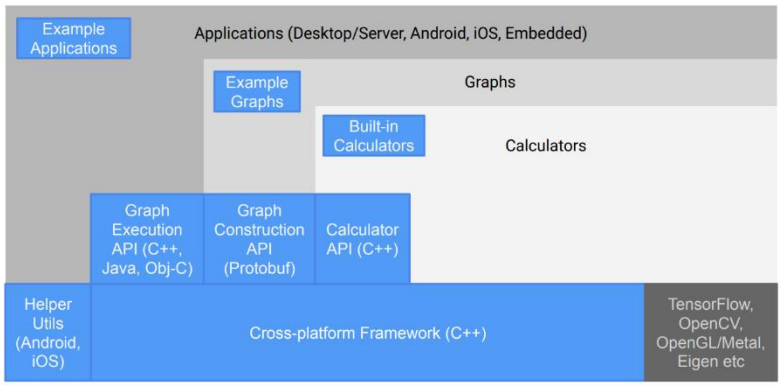
本项目使用了 MediaPipe 中的 MediaPipe Hands 模型以及 python 的 AutoPy GUI 工具包。

3.2 基本原理与使用

3.2.1 MediaPipe Hands

1. MediaPipe 介绍

MediaPipe 是一款由 Google Research 开发并开源的多媒体机器学习模型应用框架。MediaPipe 的核心框架由 C++实现。MediaPipe 的主要概念包括数据包（Packet）、数据流（Stream）、计算单元（Calculator）、图（Graph）以及子图（Subgraph）。数据包是最基础的数据单位，一个数据包代表了在某一特定时间节点的数据，例如一帧图像或一小段音频信号；数据流是由按时间顺序升序排列的多个数据包组成，一个数据流的某一特定时间戳（Timestamp）只允许至多一个数据包的存在；而数据流则是在多个计算单元构成的图中流动。MediaPipe 的图是有向的——数据包从数据源（Source Calculator 或者 Graph Input Stream）流入图直至在汇聚结点（Sink Calculator 或者 Graph Output Stream） 离开。



2. MediaPipe Hands 模型

MediaPipe Hands 为 MediaPipe 中的手部识别模型，它是一种高保真手和手指跟踪解决方案，采用机器学习 (ML) 从单个帧中推断出手的 21 个 3D 地标。

(1) Palm Detection Model（手掌检测模型）

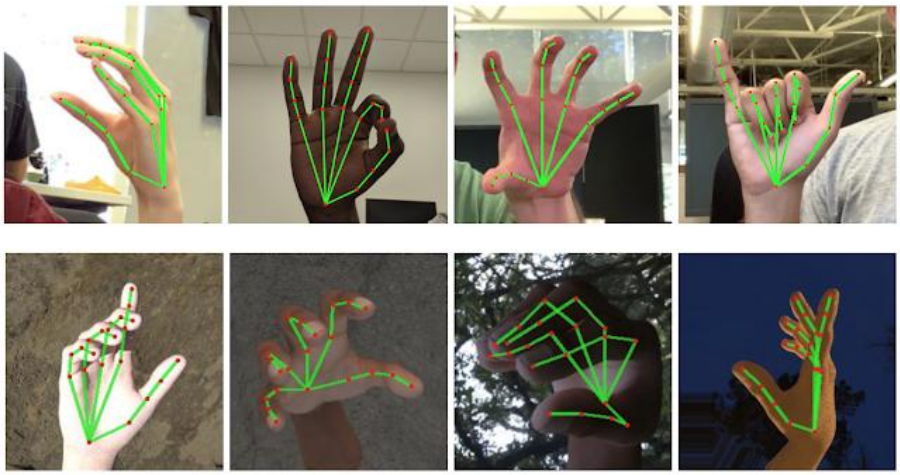
手掌检测模型，对整个图像进行操作并返回一个定向的手部边界框。

使用非极大值抑制(NMS)算法来解决手部自遮挡的问题，使用 bounding box 来建模。

(2)Hand Landmark Model（手部地标模型）

手部地标模型，对手掌检测器定义的裁剪图像区域进行操作，并返回高保真 3D 手部关键点。

在训练该模型时，使用人工注释的大约 30K 个具有 21 个 3D 坐标的真实世界图像对模型进行训练。



(3)配置选项

① STATIC_IMAGE_MODE

静态图像模式。当 static_image_mode 值设置为 false 时，则将输入图像视为视频流。它将尝试在第一个输入图像中检测手，并在成功检测后进一步定位手的界标。在随后的图像中，一旦检测到所有 max_num_hands 手并定位了相应的手标志，它就会简单地跟踪这些标志而不调用其他检测，直到它丢失任何手的跟踪。这减少了延迟，是处理视频帧的理想选择。如果设置为 true，则在每个输入图像上运行手部检测，非常适合处理一批静态的、可能不相关的图像。默认为 false。

②MAX_NUM_HANDS

要检测的最大手数，默认为 2。

③ MODEL_COMPLEXITY

手部地标模型的复杂度：0 或 1。地标准确性以及推理延迟通常随模型复杂性而增加。默认为 1.

④MIN_DETECTION_CONFIDENCE

手掌检测模型的最小置信值，用于将检测视为成功。范围为 0-1。默认为 0.5。

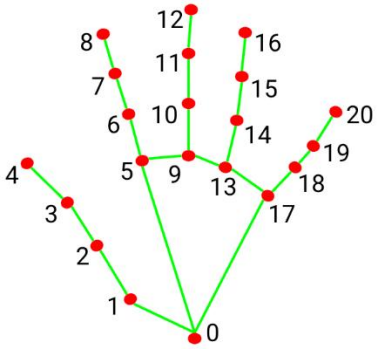
⑤MIN_TRACKING_CONFIDENCE

手部地标模型的最小置信值，将其设置为更高的值可以提高解决方案的稳健性，但代价是更高的延迟。默认为 0.5。

(4)输出

①MULTI_HAND_LANDMARKS

检测手的集合，其中每只手表示为 21 个手部标志的列表，每个标志由 x、y 和 z 组成。x 和 y 的范围由图像宽度和高度进行归一化处理为 0-1。z 表示以手腕深度为原点的地标深度，值越小，地标离相机越近。



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

②MULTI_HANDEDNESS

检测到为左手还是右手的集合。每只手都由 label 和 score 组成。label 是一个字符串"Left"或"Right"，即代表检测到的是左手还是右手。score 是预测偏手性的估计概率，并且总是大于或等于 0.5（相反的偏手性的估计概率为 1 - score）。

3. MediaPipe Hands 模型使用

(1)检测并画出手部

调用 MediaPipe Hands 模型，通过 MULTI_HAND_LANDMARKS 值画出 21 个手部标志位，并将每个手指的各部分、以及四指的底端进行连接，画出手掌的大致雏形。

(2) 判断手指是否伸出

①大拇指

对于大拇指，比较 4 点位 x 坐标与 3 点位 x 坐标的大小。若为左手，4 点位 x 坐标大于 3 点位 x 坐标即视为大拇指伸出；若为右手，4 点位 x 坐标小于 3 点位 x 坐标即视为大拇指伸出。

② 四指

对于四指，比较每一个指最上方关节点位与次上方关节点位 y 坐标的大小。若某指最上方关节点位的 y 坐标大于次上方关节点位的 y 坐标，则视为某指伸出；否则，视为某指未伸出。

3.2.3 AutoPy

1. 简介

   AutoPy 是一个简单跨平台的 Python GUI 工具包,可以控制鼠标,键盘,匹配颜色和屏幕上的位图。

2. 本项目主要使用的 autopy 方法

   (1) autopy.mouse.move()

   通过 x，y 方向的坐标控制鼠标的移动。

   (2) autopy.mouse.toggle(None, True/False)

   控制鼠标是否点击。值为 True 时控制鼠标点击，值为 False 时控制鼠标松开，默认为鼠标左键。

   (3) autopy.mouse.click(autopy.mouse.Button.RIGHT)

   控制鼠标右键的点击。

   (4) autopy.key.toggle(autopy.key.Code.DOWN_ARROW, True/False, [])

   控制键盘向下箭头的键入。

   (5) autopy.key.toggle(autopy.key.Code.UP_ARROW, True/False, [])

   控制键盘向上箭头的键入。

4 方案设计

4.1 最后采用的方案的理由：

1、所采用的为 mediapipe、opencv 和 autopy 实现的人机交互-手势控制，其中实现交互控制的检测模型为轻量化模型，在手的数量较小的时候，可以实现高时效性的检测和达到较高识别成功率。并且 autopy 可以实现较多功能的简单的跨平台 GUI 自动化工具包，实现键盘鼠标的控制和屏幕的显示。

2、如果自己训练模型，需要大约 3000k 张图片，对于训练要求较高，并且效果不太稳定，所以我们需要已经训练好的轻量化模型作为基础。

3、本方法采用 mediapipe 实现的人机交互-手势控制，其中可以直接调用其 API 完成目标检测、人脸检测以及关键点检测等，我们这里使用它进行手部的识别。其中将用 21 个点位标记我们的手部，并记录每个点位的坐标与数据，然后我们可以利用每个点位的相对位置

去对手部的动作进行一定的判断，然后识别出这个状态的手是什么手势，最后用 autopy 对鼠标和键盘进行控制，最终达成我们人机交互的目的。

4.2 本方案的主要思路

4.2.1 手部的检测和数据记录

其中我们首先利用 opencv 获取我们摄像头截取的图片，之后将图片经行一定的处理，再用训练好的模型进行检测，用 mediapipe 对我们的手部进行一个判定，并且记录我们的手的标签，类别以及各个点位的位置数据，之后我们就要对手的的位置经行处理，计算出手指在屏幕中所映射的位置信息，以方便我们之后用手指位置控制 pc 端的运行，判断完位置之后，我们便要对每个手指的情况进行判定，以方便我们对整个手的动作的一个判定，我们利用这个 21 个点位的相对位置，判断每个手指的抬起和放下，并且存放记录该数据，到此检测和判断完毕。

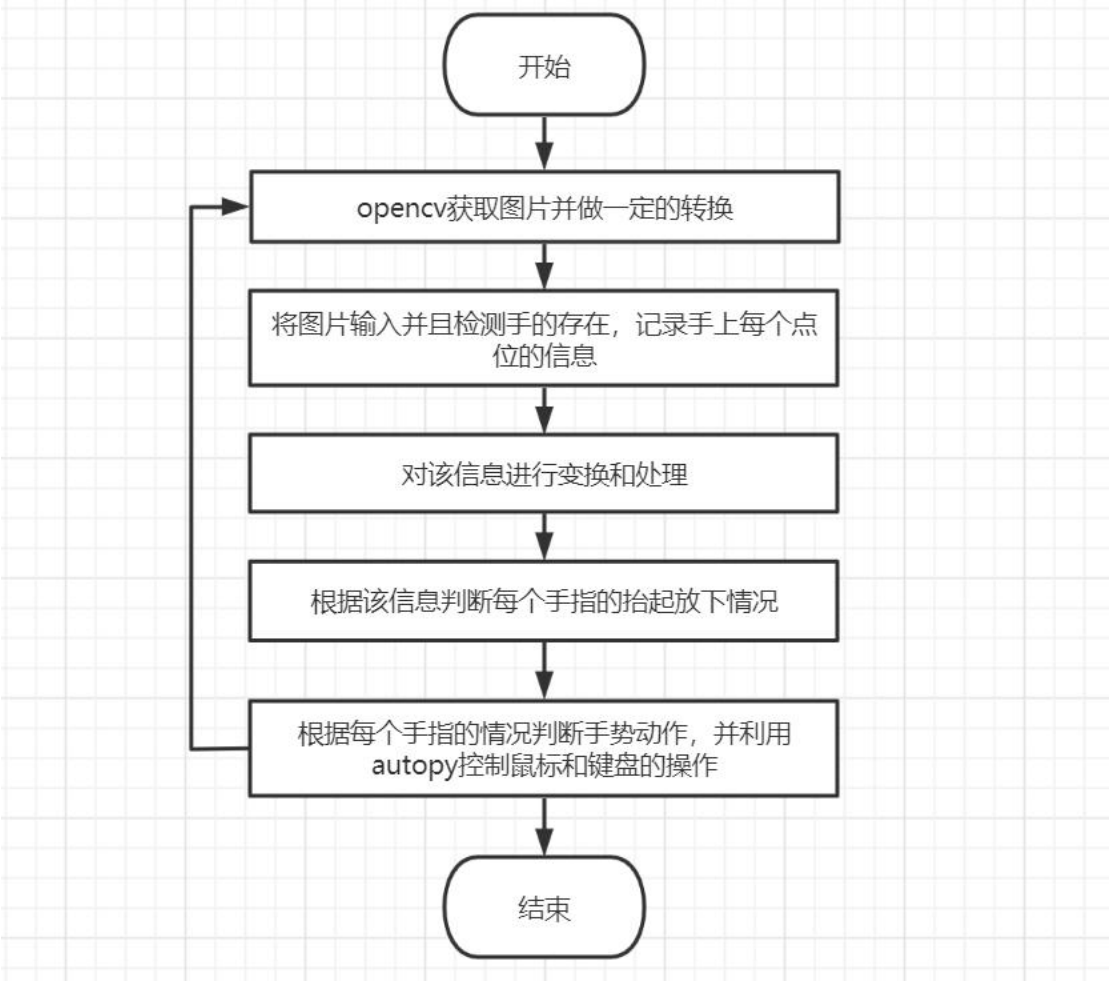
4.2.2 实现控制：

这里我们首先得到了上述所说的 21 个手的位置信息，并且得到了每个手指的抬起和放下的信息。在这之后我们边对每个手指的抬起放下做出排列组合，最终判断该手势是否为我们所需要的手势，若为该手势，则我们便可以利用 autopy 这个 GUI 工具包对我们的鼠标价盘进行控制，并且将我们的手的图形反映出来，并在屏幕上显示每个点位的信息，以方便我们更加简单的操作。

4.3 本方案的主要框图

框图如下：

4.3.1 主程序框图



实现方法：

```
def findHands(self, img, draw=True)

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

def fingersUp(self)

def findDistance(self, p1, p2, img, draw=True, r=15, t=3)
```

其中：

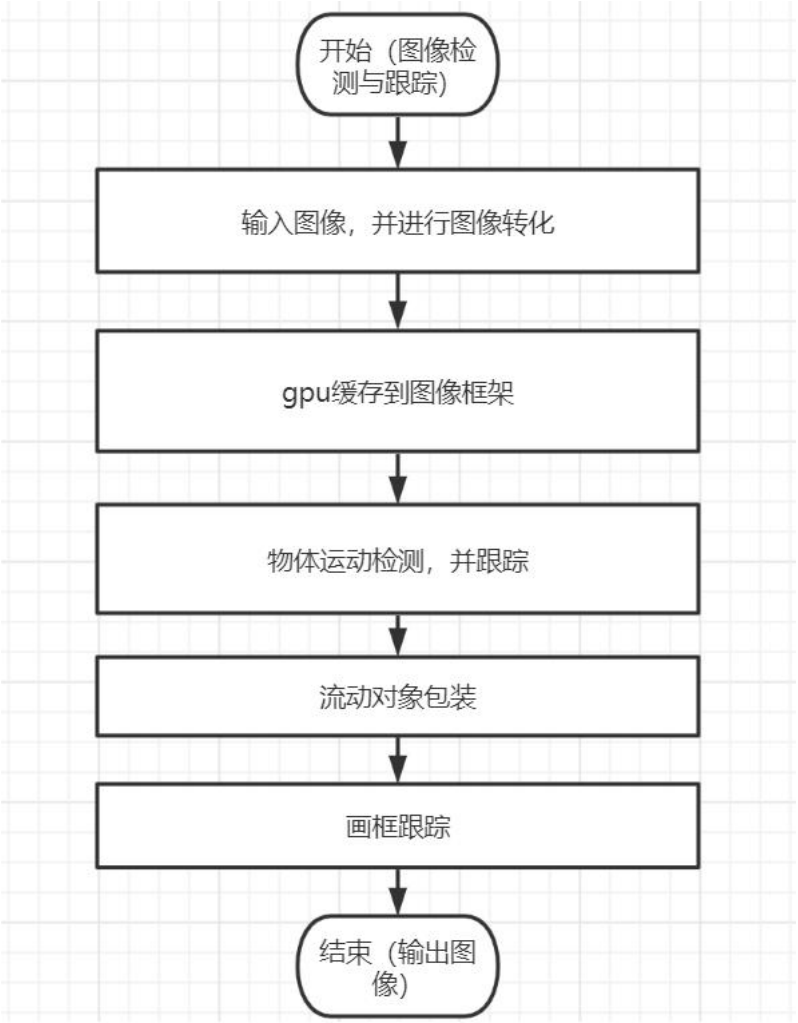
第一个函数可以检测手的存在，并且给出 21 个点位的原始数据。

第二个函数可以将 21 个点位的数据进行变化，最总得到在显示屏上映射的相对位置信息。

第三个函数可以根据节点的相对位置判断每每根手指的抬起和放下情况。

第四个函数可以得到两根手指末端的间距大小。

4.3.2 图像检测和跟踪框图



对象检测和跟踪管道可以实现为一个 MediaPipe 图，它内部利用了一个对象检测子图、一个对象跟踪子图和一个渲染子图。

一般来说，对象检测子图(在内部执行 ML 模型推断)仅在请求时运行，例如以任意帧速率或由特定信号触发。更具体地说，在这个特殊的图中，一个 PacketResampler 计算单元在传入的视频帧被传递到对象检测子图之前，暂时对其进行 0.5 fps 的子采样。这个帧率可以配置为不同的选项在 PacketResampler。

目标跟踪子图在每一帧上实时运行，跟踪被检测到的目标。它扩展了盒跟踪子图，增加了额外的功能:当新的检测到达时，它使用 IoU(交叉 Union)将当前跟踪的对象/盒与新的检测相关联，以删除过时或重复的目标框。

5 详细设计

5.1 模型框架

5.1.1 框架来源

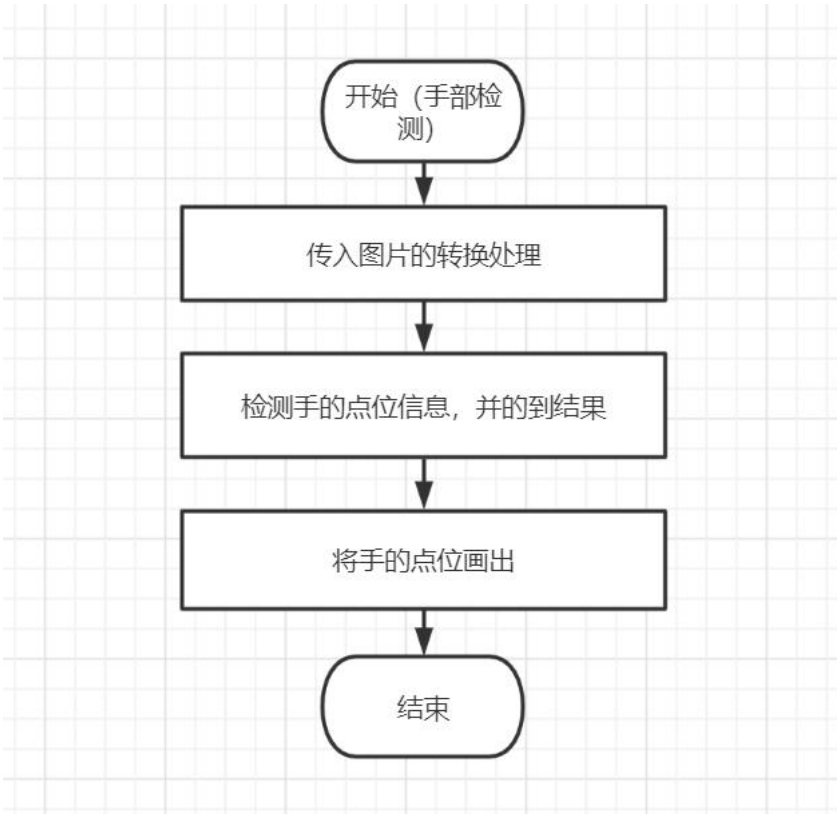
本系统使用的框架来源于Google Research 开发并开源的多媒体机器学习模型应用框架，可以直接调用其 API 完成目标检测、人脸检测以及关键点检测等。

5.1.2 框架的使用

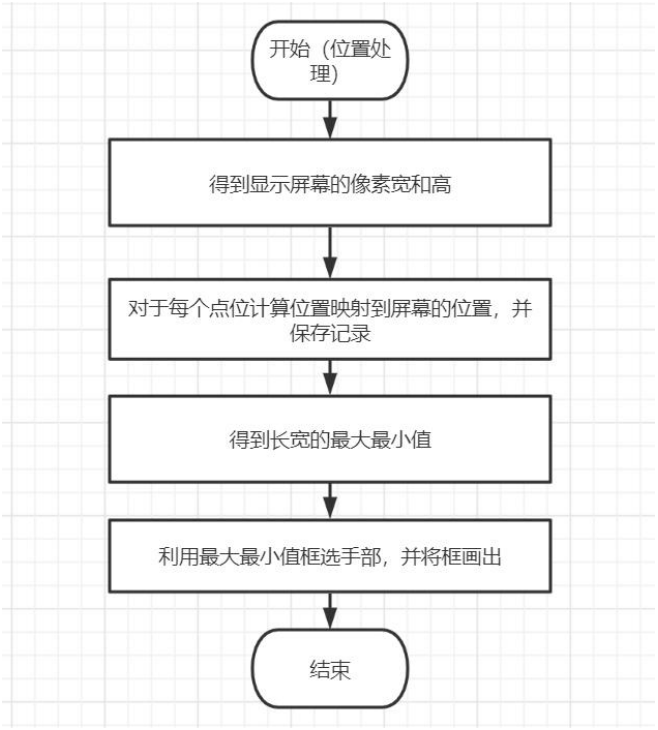
这个我们小组利用该框架实现手势的识别功能，获取检测后的 21 个点位的信息，只用利用该点位的信息实现手势的判定和人机的交互功能。

5.2 各模块具体实现流程

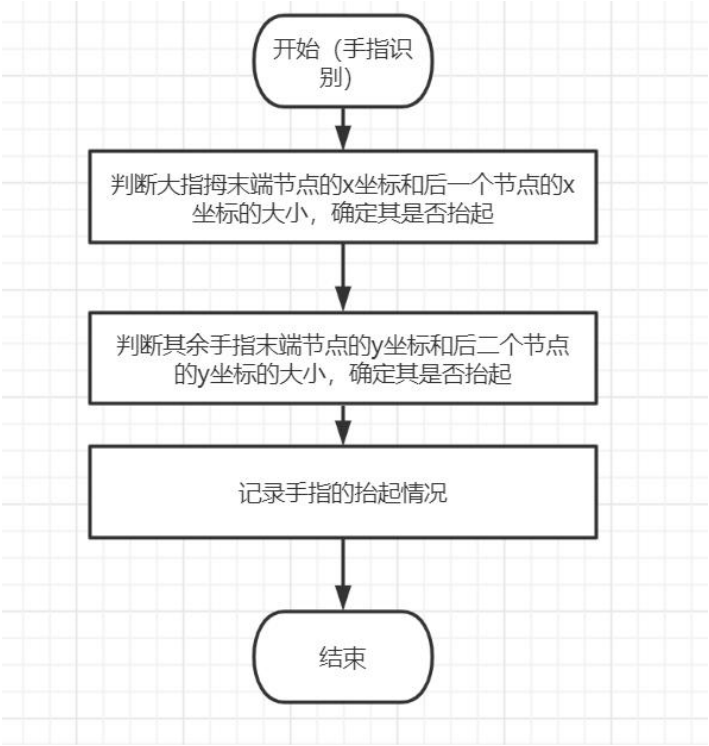
5.2.1 手部检测模块：



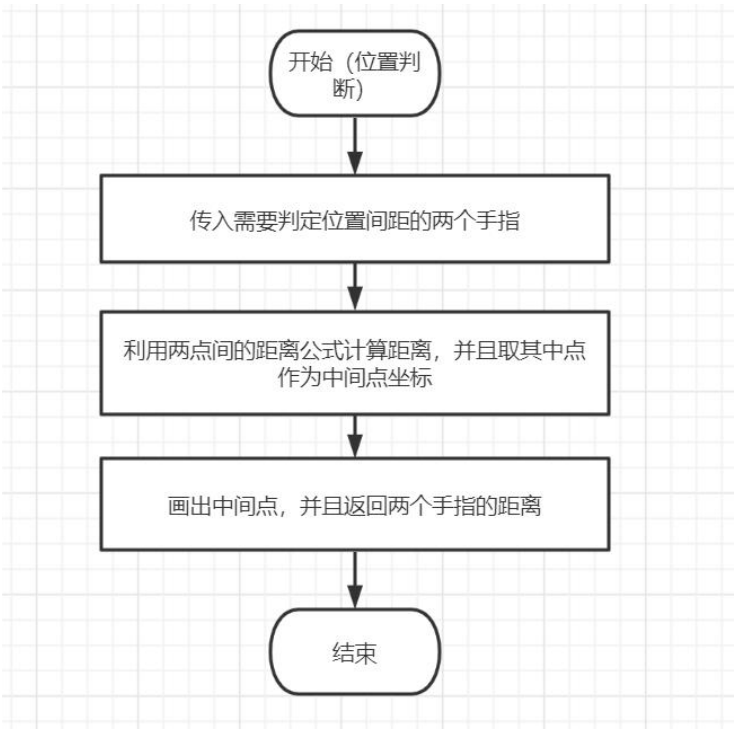
5.2.2 位置处理模块：



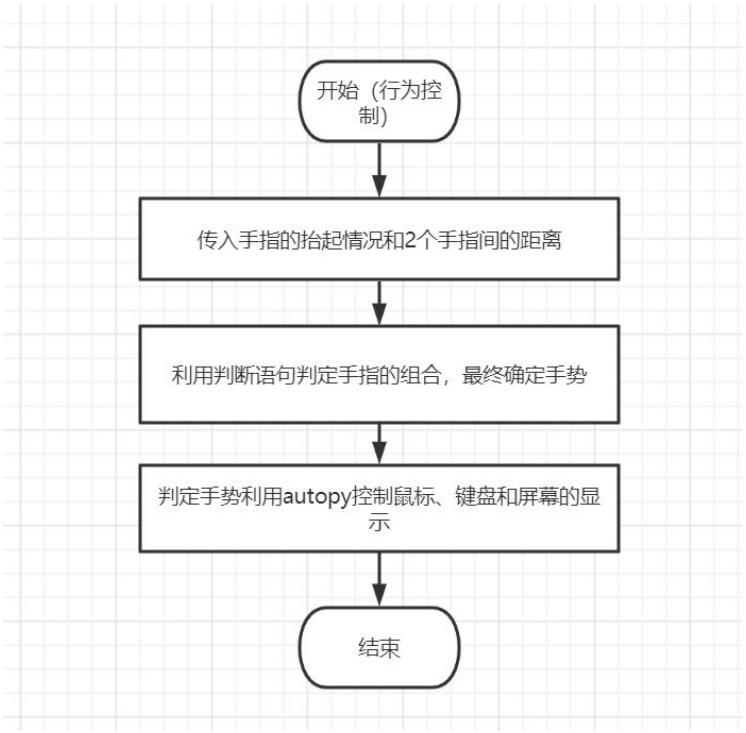
5.2.3 手指识别模块：



5.2.4 距离判断模块



5.2.5 行为控制模块



5.3 对硬件的考虑

采用的为笔记本电脑进行的系统实现。其中：

处理器为：

Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz　1.19 GHz

显卡为：

NVIDIA GeForce MX350

对于机器训练识别数字，这些固件完全可以胜任

内存为：16G 硬盘存贮为：512G 的固态硬盘

对于轻量化识别手势，并且进行人机交互完全够用。

因此对于硬件的考虑完全足够实手势识别控制的人机交互。

6 系统实现

6.1 开发环境等描述：

利用 python 环境进行系统实现
控制鼠标键盘，并进行人机交互主程序需要利用以下包：

```
import cv2
    import numpy as np
    import HandTrackingModule as htm
    import time
    import autopy
```

手势识别，并得到信息利用以下包

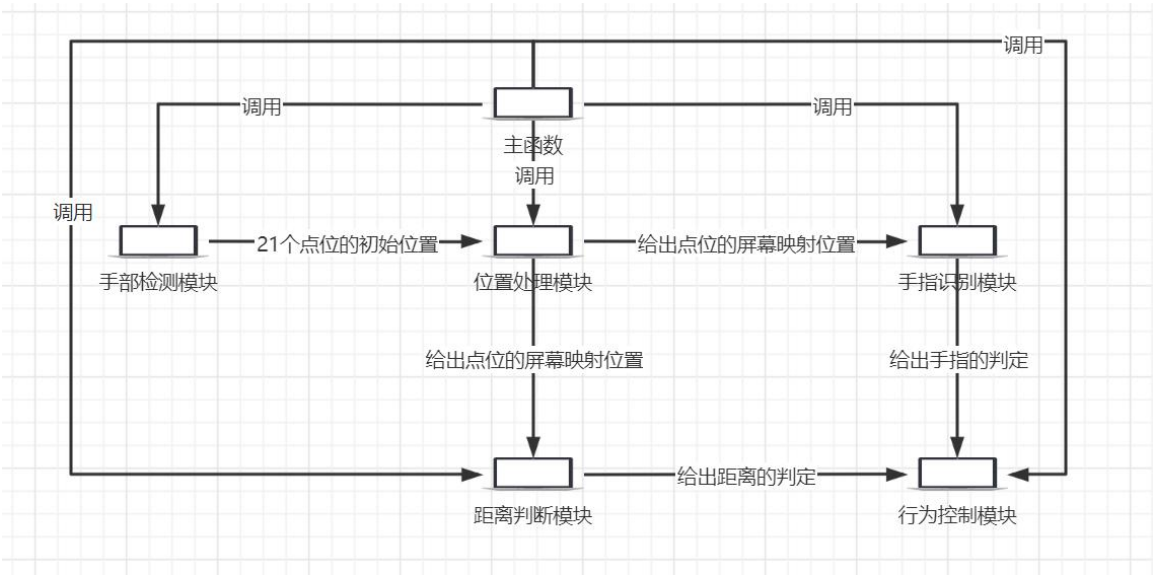
```
import cv2
    import mediapipe as mp
    import time
    import math
```

6.2 代码结构和调用关系：

识别代码被分成了 5 个模块，分别为手部检测模块，位置处理模块，手指识别模块，距离判断模块，行为控制模块。分别实现，手部检测，位置处理，手指识别，距离判断，行为控制的功能。

其中的调用关系为这 5 个模块都在主函数中被调用，然后各个模块给出对应需要的数据。

如下图：



各模块的对应关系：

手部识别理模块会利用所使用的框架将图片中的手识别，并给出 21 个点位信息给位置处理模块。

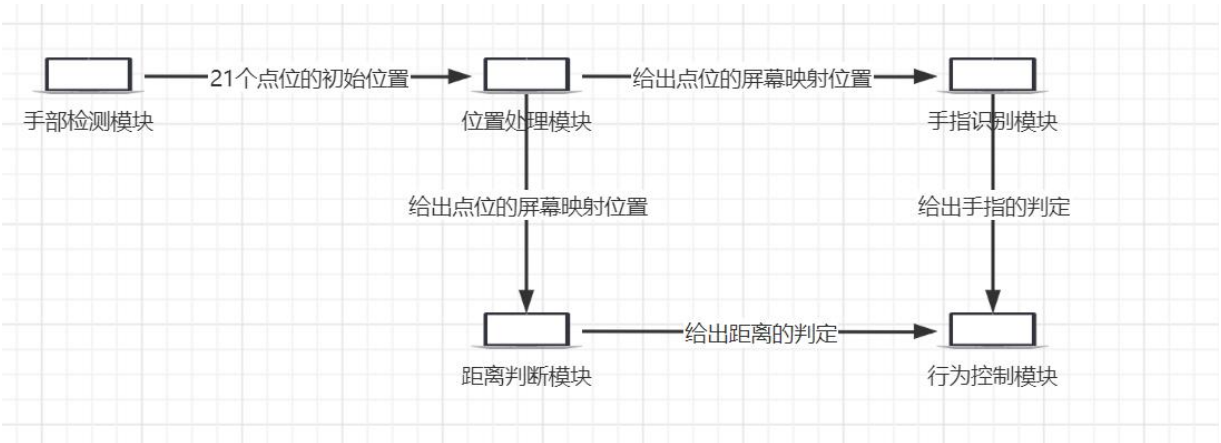
位置处理模块会利用 21 个点位的信息我，算出屏幕映射位置，并传给手指识别模块和距离判断模块。

之后手指识别模块会利用映射位置判断手指的抬起判断结果，最终返回给行为控制模块。

距离判断模块会利用映射位置算出两个手指间的距离，并返回结果给行为控制模块。、

最终行为控制模块将根据判断，控制鼠标键盘，以及屏幕的显示。

关系如下图：



6.3 代码展示

6.3.1 手部检测和人机交互控制主程序代码

```
import cv2
import numpy as np
import HandTrackingModule as htm
import time
import autopy

##########################
wCam, hCam = 640, 480
frameR = 100  # Frame Reduction
smoothening = 7
#########################

pTime = 0
plocX, plocY = 0, 0
clocX, clocY = 0, 0

cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
detector = htm.handDetector(maxHands=1)
wScr, hScr = autopy.screen.size()
# print(wScr, hScr)

flag = 0
while True:
    # 1. Find hand Landmarks
    success, img = cap.read()
    img = detector.findHands(img)
    lmList, bbox = detector.findPosition(img)
    # 2. Get the tip of the index and middle fingers
    if len(lmList) != 0:
        x1, y1 = lmList[8][1:]
        x2, y2 = lmList[12][1:]
        # print(x1, y1, x2, y2)

    cv2.rectangle(img, (frameR, frameR), (wCam - frameR, hCam - frameR),
                  (255, 0, 255), 2)
    # 3. Check which fingers are up
    fingers = detector.fingersUp()
```

```
    if fingers:
        # print(fingers)
        # 4. Only Index Finger : Moving Mode
        if fingers[1] == 1 and fingers[2] == 0:
            # 5. Convert Coordinates
            x3 = np.interp(x1, (frameR, wCam - frameR), (0, wScr))
            y3 = np.interp(y1, (frameR, hCam - frameR), (0, hScr))
            # 6. Smoothen Values
            clocX = plocX + (x3 - plocX) / smoothening
            clocY = plocY + (y3 - plocY) / smoothening

            # 7. Move Mouse
            autopy.mouse.move(wScr - clocX, clocY)
            cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
            plocX, plocY = clocX, clocY

        # 8. Both Index and middle fingers are up : Clicking Mode
        if fingers[1] == 1 and fingers[2] == 1:
            # 9. Find distance between fingers
            length, img, lineInfo = detector.findDistance(8, 12, img)
            if length < 40 and flag == 0:
                cv2.circle(img, (lineInfo[4], lineInfo[5]),
                           15, (0, 255, 0), cv2.FILLED)
                autopy.mouse.toggle(None,True)
                flag = 1
            elif length >= 40 and flag == 1:
                cv2.circle(img, (lineInfo[4], lineInfo[5]),
                           15, (0, 255, 0), cv2.FILLED)
                autopy.mouse.toggle(None,False)
                flag = 0

        if fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 0:
            autopy.mouse.click(autopy.mouse.Button.RIGHT)

        if fingers[0] == 1 and fingers[1] == 0 and fingers[2] == 0:
            autopy.key.toggle(autopy.key.Code.DOWN_ARROW, True, [])
            autopy.key.toggle(autopy.key.Code.DOWN_ARROW, False, [])

        if fingers[0] == 1 and fingers[1] == 1 and fingers[2] == 1:
            autopy.key.toggle(autopy.key.Code.UP_ARROW, True, [])
            autopy.key.toggle(autopy.key.Code.UP_ARROW, False, [])

            # 11. Frame Rate
    cTime = time.time()
```

```
        fps = 1 / (cTime - pTime)
        pTime = cTime
        cv2.putText(img, str(int(fps)), (20, 50), cv2.FONT_HERSHEY_PLAIN, 3,
                                    (255, 0, 0), 3)
        # 12. Display
        cv2.imshow("Image", img)
        cv2.waitKey(1)
```

6.3.2 手部识别跟踪代码

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


class handDetector():
    def __init__(self, mode=False, maxHands=1, 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)
        #print(self.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_CONNECTIONS)

        return img

```

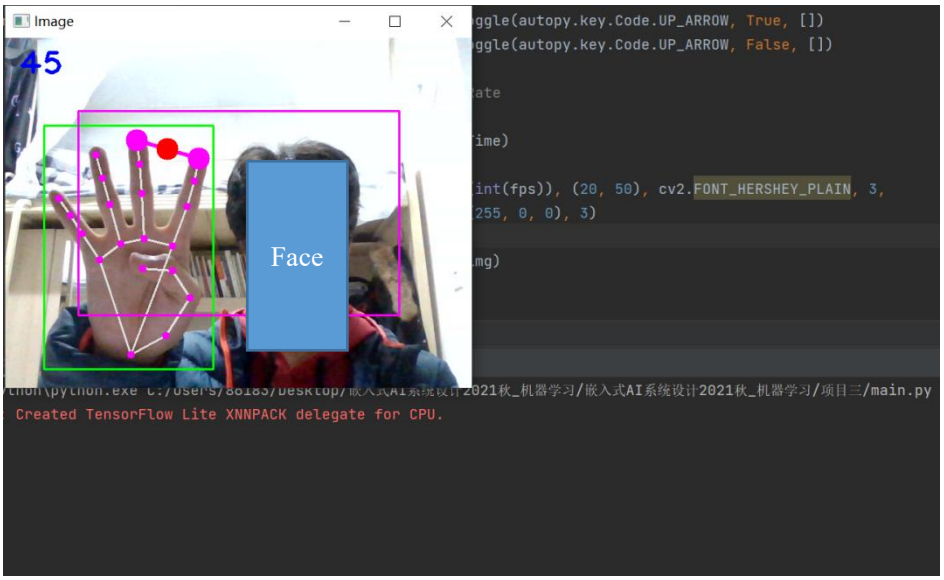
```
34.     def findPosition(self, img, handNo=0, draw=True):
35.         xList = []
36.         yList = []
37.         bbox = []
38.         self.lmList = []
39.         if self.results.multi_hand_landmarks:
40.             myHand = self.results.multi_hand_landmarks[handNo]
41.             for id, lm in enumerate(myHand.landmark):
42.                 # print(id, lm)
43.                 h, w, c = img.shape
44.                 cx, cy = int(lm.x * w), int(lm.y * h)
45.                 xList.append(cx)
46.                 yList.append(cy)
47.                 self.lmList.append([id, cx, cy])
48.                 if draw:
49.                     cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)
50.
51.             xmin, xmax = min(xList), max(xList)
52.             ymin, ymax = min(yList), max(yList)
53.             bbox = xmin, ymin, xmax, ymax
54.
55.             if draw:
56.                 cv2.rectangle(img, (xmin - 20, ymin - 20), (xmax + 20, ymax +
    20),
57.                               (0, 255, 0), 2)
58.
59.         return self.lmList, bbox
60.
61.     def fingersUp(self):
62.         fingers = []
63.         if self.lmList:
64.             if self.lmList[self.tipIds[0]][1] > self.lmList[self.tipIds[0] - 1][1] \
65.                 and self.lmList[self.tipIds[1]][1] > self.lmList[self.tipIds[2]][1] \
66.                 or self.lmList[self.tipIds[0]][1] < self.lmList[self.tipIds[0] - 1][1] \
67.                 and self.lmList[self.tipIds[1]][1] < self.lmList[self.tipIds[2]][1]:
68.                 fingers.append(1)
69.             else:
70.                 fingers.append(0)
71.
72.             # Fingers
73.             for id in range(1, 5):
74.
75.                 if self.lmList[self.tipIds[id]][2] < self.lmList[self.tipIds[id] - 2][2]:
76.                     fingers.append(1)
```

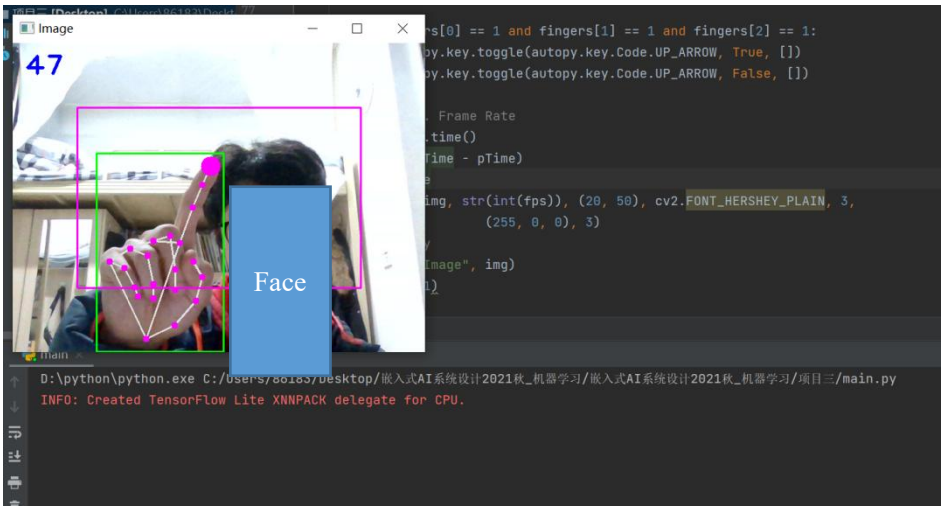
```
77.                     else:
78.                         fingers.append(0)
79.
80.         # totalFingers = fingers.count(1)
81.
82.         return fingers
83.
84.     def findDistance(self, p1, p2, img, draw=True, r=15, t=3):
85.         x1, y1 = self.lmList[p1][1:]
86.         x2, y2 = self.lmList[p2][1:]
87.         cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
88.
89.         if draw:
90.             cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), t)
91.             cv2.circle(img, (x1, y1), r, (255, 0, 255), cv2.FILLED)
92.             cv2.circle(img, (x2, y2), r, (255, 0, 255), cv2.FILLED)
93.             cv2.circle(img, (cx, cy), r, (0, 0, 255), cv2.FILLED)
94.         length = math.hypot(x2 - x1, y2 - y1)
95.
96.         return length, img, [x1, y1, x2, y2, cx, cy]
97.
98.
99. def main():
100.     pTime = 0
101.     cTime = 0
102.     cap = cv2.VideoCapture(0)
103.     detector = handDetector()
104.     while True:
105.         success, img = cap.read()
106.         img = detector.findHands(img)
107.         lmList, bbox = detector.findPosition(img)
108.         if len(lmList) != 0:
109.             print(lmList[4])
110.
111.         cTime = time.time()
112.         fps = 1 / (cTime - pTime)
113.         pTime = cTime
114.
115.         cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3,
116.                     (255, 0, 255), 3)
117.
118.         cv2.imshow("Image", img)
119.         cv2.waitKey(1)
120.
```

```
121.
122.  if __name__ == "__main__":
123.      main()
```

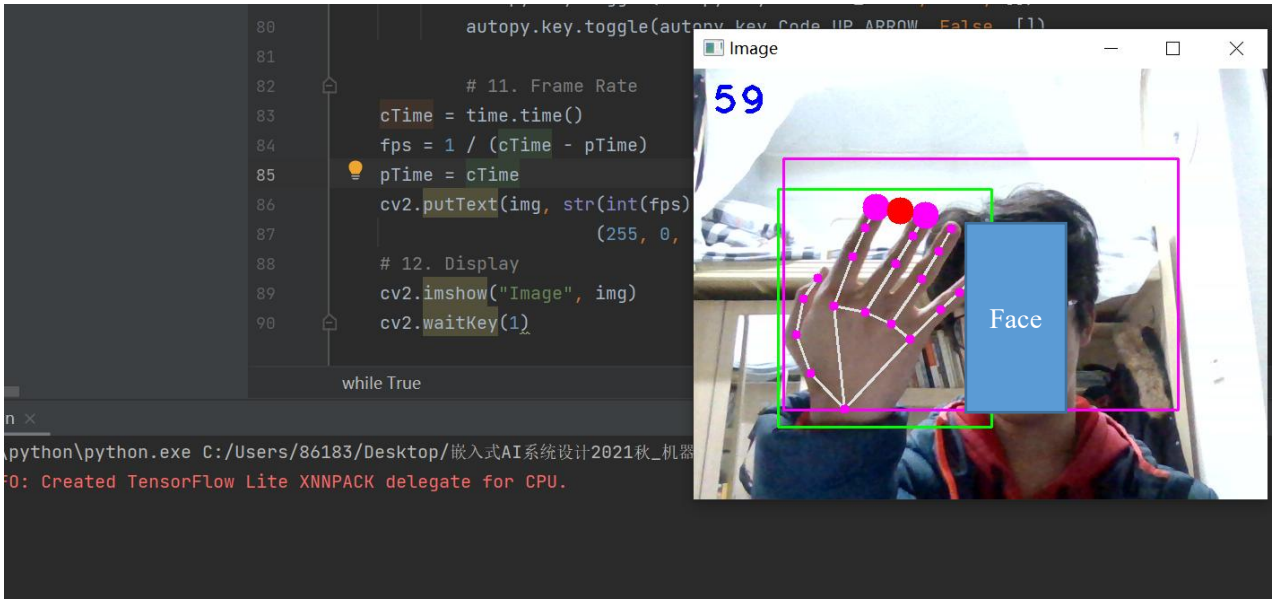
7 测试及评估结果

给出多组测试及评估结果，并适当分析。





分析：因为利用轻量级的识别，可以实现较高帧率的显示，对于手的识别也成功率十分高，无论实在或明或暗，或者将手藏起来部分，也可以识别出来，并且显示点位，并且对于手势控制鼠标和键盘十分成功，并不会出现太明显的误判和未识别。交互控制过程很流畅，没有出现滞留和延时的情况，但对于手翻转后的手势判定，无法识别，于是我们利用位置信息进行判断，实现手翻面后的识别和控制。



分析：再加入判断后，右手反转或者左手控制也可以实现相对应的功能，没有出现控制判断错误的情况产生，并且，整个人机交互过程也十分的流畅和高效。

8 总结与对于新的人机交互方式的畅想

1.总结：

本次课程设计对于刚刚步入大二的我们来说着实是一个不小的挑战，在此之前，“人机交互”这四个大字让我们感到“畏途巉岩不可攀”。

经过一次次讨论，最终我们决定我们要做手势控制。我们发现 Google 公司的 Mediapipe 能给我们很大的发挥空间，其 Mediapipe hands 模型为轻量化模型，适合之后将其迁移到树莓派上实现。至于如何实现鼠标点击、移动操作以及键盘的快捷键功能，我们则选用了 autopy 库，这为项目的实现提供了些便利

这一过程也不是一帆风顺，面对代码实现中一行行红色的 error 我们也显得束手无策，面对鼠标连点、误触我们也显得措手不及，面对无法区分左右手的情况我们也苦苦思索解决之道。但我们一次次地分工学习、一次次地小组讨论，虽然过程磕磕绊绊，但最终还是实现了这次的课程设计。

2.对于未来新的人机交互方式的畅想：

如今，人机交互方式早已不是“旧时王谢堂前燕”，手势控制、语音控制、亦或是苹果 imac 上的黑科技眼球控制等等，已经“飞入寻常百姓家”。除此之外，各种穿戴传感器与

脑电波控制方式也在紧锣密鼓地更新迭代。

但我们不难发现，以上所有的人机交互方式都是“人”影响“机”、“人”控制“机”，发出指令的往往是人，而接受指令执行操作的往往是机器。可是“交互”一词的含义是双向的，仅仅单方面的影响难以算得上“交流与互动”。我们在设想，未来有没有一种可能，能让机器更多地影响人的生活与生产；有没有可能，让机器识别我们的情感——也就是实现情感交互。

众所周知，在如今快节奏、高压力的社会环境下，越来越多的人有了心理方面的问题，若是机器能实时识别我们的情绪，对我们进行提前的心理疏导与干预，我想这样会对年轻人的心理健康大有裨益。

同时，情感交互也能用于解决目前十分火热的“空巢老人”问题。目前所有的解决方案都更倾向于对老年人的“物理关怀”，比如帮老人拿东西、做事情，监控老人是否跌倒，且大多情况下老人与机器的人机交互是单方向的，是“主仆”关系；但我们更应该关注空巢老人的精神关怀，若是机器能具有情感，真正具有“陪伴”功能，可以主动和老人进行交流，排解老人孤独寂寞的心理，具有“共情心”，把“主仆关系”转化为“朋友关系”甚至是“家人关系”，这种人机交互的方式才真正算的上是“交互”。

正所谓“夏虫不可语冰”，上世纪初的人们以为未来的交通方式是横穿在高楼大厦天际间的眼花缭乱的快速轨道，谁也未曾设想如今我们早已有了更加合理的解决方式——地下铁路。我相信未来的人机交互方式无论形式如何，一定会更加合理、方便、功能强大，给人们的生活带来更多的便利与美好。