

Image Browser Based on Gesture Recognition

Team member	task
2152057 杨瑞华	Core code logic implementation
2152038 戈雨阳	Core code logic implementation
2152831 陈峥海	documentation and PowerPoint

1. Background

1.1 Environment

compiler: Python3.7+

package needed: opencv-python, MediaPipe, PyQt5

Use `opencv` to complete the camera video stream reading, use `mediapipe` to build a hand joint point recognition model, use `pyqt5` to build a picture browser `gui` interface.

1.2 How to run

1. Install required modules

```
1 | pip install pyqt5
2 | pip install opencv-python
3 | pip install mediapipe
```

2. Run python script

```
1 | python ./picture_viewer.py
```



1.3 Idea

We consider that in some scenarios, people may not be able to directly click on devices such as mobile phones or computers for device control. For example, watching a projection screen TV in bed or watching the recipe process while cooking, and not using a mobile phone if you have oil stains on your hands. So we want to do a gesture interaction to control the device through different hand movements. Here, we are controlling the album.

2. Description

2.1 Overview

This project realizes the album interaction based on Gesture recognition.

User can perform different operations on the album based on different gestures.

2.2 Structures

The following is the core structure of the code——

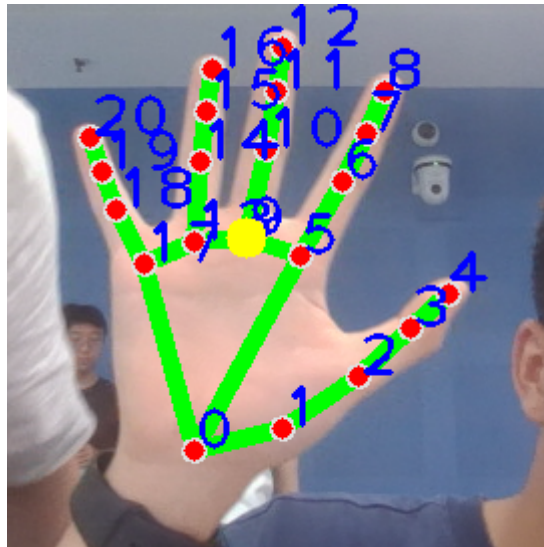
```
.
| fingersVector.py
| GestureRecognition.py
| HandLandmarks.py
| picture_viewer.py
| picture_viewer.ui
| picture_viewer_ui.py
|
└─picture
```

The picture folder contains some default images.

The resource folder contains some interactive images.

2.3 Code Logic

- `HandLandmarks.py` calls the `mediapipe` package to recognize the hand model based on machine learning and obtain the coordinates of the 21 joint point models of the hand



```
1  import mediapipe as mp
2  import cv2 as cv
3
4  # 绘制关键点与连接线函数
5  mp_drawing = mp.solutions.drawing_utils
6  handMsStyle = mp_drawing.DrawingSpec(
7      color=(0, 0, 255), thickness=int(5)) # 关键点样式
8  handConStyle = mp_drawing.DrawingSpec(
9      color=(0, 255, 0), thickness=int(8)) # 关键点连接线样式
10 # 手部检测函数
11 mp_hands = mp.solutions.hands
12 hands = mp_hands.Hands(
13     static_image_mode=False, # 检测的是视频流还是静态图片, False为视频流,
    True为图片
14     max_num_hands=1, # 检测出手的最大数量
15     min_detection_confidence=0.75, # 手部检测的最小置信度, 大于该值则认为检
    测成功
16     min_tracking_confidence=0.75) # 目标跟踪模型的最小置信度
17
18
19 def handLandmarks(frame):
```

```

20     imgRGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
21     result = hands.process(imgRGB)
22     hand_point = result.multi_hand_landmarks # 返回21个手部关键点的坐
标, 其值为比例
23     # result = hands.process(imgRGB)
24     # hand_point = result.multi_hand_landmarks # 返回21个手部关键点的坐
标, 其值为比例
25
26     landmarks = []
27     # 当关键点存在时
28     if hand_point:
29         for handlms in hand_point:
30             # print(handlms)
31             # 绘制关键点及其连接线, 参数:
32             # 绘制的图片, 关键点坐标, 连接线, 点样式, 线样式
33             mp_drawing.draw_landmarks(
34                 frame, handlms, mp_hands.HAND_CONNECTIONS,
handMsStyle, handConStyle)
35             for i, lm in enumerate(handlms.landmark):
36                 # lm.x表示在图片大小下的比例, 乘以图片大小将其转换为队形坐标
37                 posX = int(lm.x * frame.shape[1])
38                 posY = int(lm.y * frame.shape[0])
39                 landmarks.append((posX, posY)) # 21个手部关键点坐标
40                 # 显示关键点
41                 cv.putText(frame, str(i), (posX, posY),
42                             cv.FONT_HERSHEY_PLAIN, 2, (255, 0, 0),
thickness=2)
43
44             return landmarks
45         else:
46             return "error"

```

- According to the coordinates of the 21 hand points obtained, calculate the connection and position setting relationship of different points in `fingersVector.py`, and get the protruding situation of each finger

```

1     import math
2
3     # 计算向量2范数
4     def vectorSize(p1,p2):
5         return math.sqrt((p1[0]-p2[0])**2+(p1[1]-p2[1])**2)
6
7     # 余弦定理计算向量夹角
8     def vectorAngle(p1,p2,p3):
9         b = math.sqrt((p2[0] - p1[0]) ** 2 + (p2[1] - p1[1]) ** 2)
10        c = math.sqrt((p2[0] - p3[0]) ** 2 + (p2[1] - p3[1]) ** 2)

```

```

11     a = math.sqrt((p3[0] - p1[0]) ** 2 + (p3[1] - p1[1]) ** 2)
12     if ((2 * b * c) > 1e-10):
13         cos = (b ** 2 + c ** 2 - a ** 2) / (2 * b * c)
14         if cos > 1:
15             cos = 1
16         elif cos < -1:
17             cos = -1
18         angle = math.acos(cos)
19     return math.degrees(angle)
20
21     # 由坐标点构造向量
22     def mkVector(p1, p2):
23         return [(p1[0] - p2[0]), (p1[1] - p2[1])]
24
25     # 计算两个向量的夹角
26     def vectorAngle2(v1, v2):
27         nor = 0.0
28         a = 0.0
29         b = 0.0
30         for x, y in zip(v1, v2):
31             nor += x * y # 向量内积
32             a += x ** 2
33             b += y ** 2
34         if a == 0 or b == 0:
35             return None
36         cosTheta = nor / math.sqrt(a * b)
37         angle = math.acos(cosTheta)
38         return math.degrees(angle)
39
40     # 判断手指是否张开
41     def fingersUp(landmarks):
42         fingers = []
43         # 大拇指
44         if vectorAngle(landmarks[0], landmarks[3], landmarks[4]) > 130:
45             fingers.append(1)
46         else:
47             fingers.append(0)
48
49         # 食指
50         if
51             vectorSize(landmarks[0], landmarks[8]) > vectorSize(landmarks[0], landmarks[6]):
52                 fingers.append(1)
53             else:
54                 fingers.append(0)

```

```

55     # 中指
56     if
vectorSize(landmarks[0],landmarks[12])>vectorSize(landmarks[0],landmarks[10]):
57         fingers.append(1)
58     else:
59         fingers.append(0)
60
61     # 无名指
62     if
vectorSize(landmarks[0],landmarks[16])>vectorSize(landmarks[0],landmarks[14]):
63         fingers.append(1)
64     else:
65         fingers.append(0)
66
67     # 小拇指
68     if
vectorSize(landmarks[0],landmarks[20])>vectorSize(landmarks[0],landmarks[18]):
69         fingers.append(1)
70     else:
71         fingers.append(0)
72
73     return fingers

```

- Different gestures are recognized according to the extension of different fingers, the position coordinates of the center of gravity of the palm in the two frames before and after, and the angles between the vectors of different joint points, and different gestures are obtained according to the returned `command`

```

1  commands = {1: "OK",
2              2: "thumb",
3              3: "fist",
4              4: "middle",
5              5: "spider",
6              6: "heart",
7              7: "hand",
8              8: "right",
9              9: "left",
10             10: "index",
11             11: "ring",
12             12: "little"}

```

```
1  from fingersVector import fingersUp, vectorSize, vectorAngle,
   mkVector, vectorAngle2
2  import cv2 as cv
3
4
5  def staticGestureRec(landmark):
6      command = 0
7      fingers = fingersUp(landmark)
8      if (fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 1 and
fingers[3] == 1 and fingers[4] == 1):
9          command = 1
10         # print("OK")
11         if (fingers[0] == 1 and fingers[1] == 0 and fingers[2] == 0 and
fingers[3] == 0 and fingers[4] == 0):
12             command = 2
13             # print("大拇指")
14             if (fingers[0] == 0 and fingers[1] == 1 and fingers[2] == 0 and
fingers[3] == 0 and fingers[4] == 0):
15                 command = 10
16                 # print("食指")
17
18             if (fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 0 and
fingers[3] == 1 and fingers[4] == 0):
19                 command = 11
20                 # print("无名指")
21
22             if (fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 0 and
fingers[3] == 0 and fingers[4] == 1):
23                 command = 12
24                 # print("小拇指")
25
26             if (fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 0 and
fingers[3] == 0 and fingers[4] == 0):
27                 command = 3
28                 # print("拳头")
29
30             if (fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 1 and
fingers[3] == 0 and fingers[4] == 0):
31                 command = 4
32                 # print("中指")
33
34             if (fingers[0] == 1 and fingers[1] == 1 and fingers[2] == 0 and
fingers[3] == 0 and fingers[4] == 1):
35                 command = 5
36                 # print("spider")
37
```

```

38     if (fingers[0] == 1 and fingers[1] == 1 and fingers[2] == 0 and
fingers[3] == 0 and fingers[4] == 0
39         and vectorSize(landmark[3], landmark[6]) < 20 and
vectorAngle(landmark[4], landmark[6], landmark[8]) < 90):
40         command = 6
41         # print("比心")
42     if (fingers[0] == 1 and fingers[1] == 1 and fingers[2] == 1 and
fingers[3] == 1 and fingers[4] == 1):
43         command = 7
44         # print("手掌")
45     return command
46
47
48 def gestureRecognition(frame, landmarks, preCenter):
49     # 静态手势识别
50     command = staticGestureRec(landmarks)
51
52     # 动态手势识别
53     curCenter = landmarks[9]
54     cv.circle(frame, curCenter, 10, (0, 255, 255), -1)
55     centerVector = mkVector(curCenter, preCenter) # 构造帧差向量
56     curPreSize = vectorSize(preCenter, curCenter) # 控制手势帧差大小区
域
57     xAxis = [1, 0]
58     angle = vectorAngle2(centerVector, xAxis) # 控制手势帧差角度区域
59     if centerVector[0] > 0 and curPreSize > 180 and angle < 30:
60         command = 8
61         print("right")
62
63     if centerVector[0] < 0 and curPreSize > 180 and angle > 150:
64         command = 9
65         print("left")
66
67     return curCenter, command

```

- Perform different operation logic in `picture_viewer.py` according to different gestures

```

1     if command != 0:
2         cv.putText(frame, commands[command], (170, 50),
3                     cv.FONT_HERSHEY_COMPLEX_SMALL, 2, (0,
0, 255), thickness=2)
4         if command != 0 and self.pictureListWidget.count() >
0 and precommand != command:
5
6             precommand = command
7

```



```

8         if command in [8, 9]:
9             flipCTime = time.time()
10            if flipCTime-flipPTime > 1.5:
11                if command == 8:
12                    if self.currentIndex > 0:
13                        self.currentIndex -= 1
14                elif command == 9:
15                    if self.currentIndex <
self.pictureListWidget.count()-1:
16                        self.currentIndex += 1
17                    print("switch")
18                    self.pictureListWidget.setCurrentRow(
19                        self.currentIndex)
20                    picturePath = os.path.join(
21                        self.dirPath,
self.currentPictureList[self.currentIndex])
22                    pixmap = QPixmap(picturePath)
23                    size = self.mainPictureView.size()
24                    pixmap = pixmap.scaled(
25                        size, QtCore.Qt.KeepAspectRatio)
26                    self.mainPictureView.setPixmap(pixmap)
27                    flipPTime = flipCTime
28
29            elif command == 6:
30                print("like")
31                self.like()
32
33            elif command == 5 or command == 12:
34                print("dislike")
35                self.dislike()

```

3. The implemented requirements

As mentioned above, our interaction system provides several basic functions — flipping, praising, and canceling.

The system has high scalability .

Our gestures include:

3.1 static gesture recognition

By calculating the vector angle and size between the coordinates of 21 joint points, calculate the extension of each finger, and then calculate different gestures.

Gesture	core code
OK	fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 1 and fingers[3] == 1 and fingers[4] == 1
thumbs	fingers[0] == 1 and fingers[1] == 0 and fingers[2] == 0 and fingers[3] == 0 and fingers[4] == 0
index finger	fingers[0] == 0 and fingers[1] == 1 and fingers[2] == 0 and fingers[3] == 0 and fingers[4] == 0
middle finger	fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 1 and fingers[3] == 0 and fingers[4] == 0
ring finger	fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 0 and fingers[3] == 1 and fingers[4] == 0
little finger	fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 0 and fingers[3] == 0 and fingers[4] == 1
fist	fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 0 and fingers[3] == 0 and fingers[4] == 0
palm	fingers[0] == 1 and fingers[1] == 1 and fingers[2] == 1 and fingers[3] == 1 and fingers[4] == 1
heart	fingers[0] == 1 and fingers[1] == 1 and fingers[2] == 0 and fingers[3] == 0 and fingers[4] == 0 and vectorSize(landmark[3], landmark[6]) < 20 and vectorAngle(landmark[4], landmark[6], landmark[8]) < 90
spider	fingers[0] == 1 and fingers[1] == 1 and fingers[2] == 0 and fingers[3] == 0 and fingers[4] == 1



3.2 dynamic gesture recognition

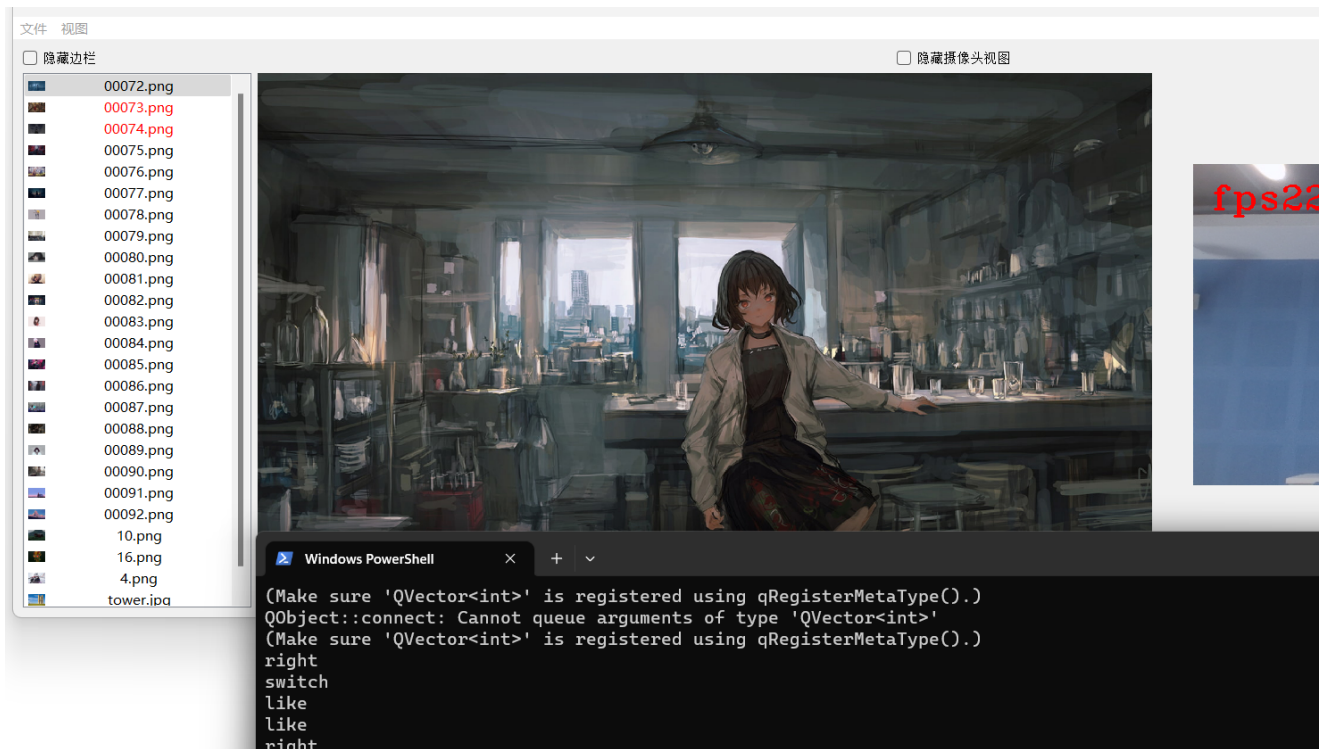
By marking the position vectors of the hand center of gravity before and after the two frames, calculating the angle between the vector and the x-direction, as well as the length of the vector, the dynamic gesture of left and right swing is recognized.

Gesture	core code
left	$\text{centerVector}[0] < 0$ and $\text{curPreSize} > 180$ and $\text{angle} > 150$
right	$\text{centerVector}[0] > 0$ and $\text{curPreSize} > 180$ and $\text{angle} < 30$

3.3 interactive functions

Our interactive functions include the following points:

gesture	interactive function
left	Flip the album to the next page
right	Flip the album to the previous one
heart	Set the current photo as a favorite
spider/ring finger	Cancel likes



4. Advantages and Disadvantages

4.1 Advantages

Intuitive

Users can easily understand and master the operation. For example, using left and right sliding gestures to achieve page flipping function, using pinch gestures to achieve deletion function, etc.

Visual feedback

It is important to provide clear visual feedback when users perform gesture operations. Users can know that their gestures are correctly captured and interpreted through animations, indicators, or other means. This can enhance user confidence and satisfaction.

Sensitivity setting

Consider providing some sensitivity setting options to meet the gesture input habits of different users. Some users may prefer quick sliding gestures, while others may prefer slow and accurate gestures. At the same time, in order to prevent the problem of dynamic Gesture recognition jumping too fast, we set a 1.5s interval between the two dynamic Gesture recognition

Multifunctional gestures

Consider incorporating more features into gesture design to provide more operational options. For example, a double click gesture can be implemented to enlarge a photo, or a rotation gesture can be used to adjust the direction of the photo. This can increase the functionality richness and user friendliness of the album application.

User Customization

Allowing users to customize gestures is an interesting design option. In this way, users can set gestures according to their preferences and habits, increasing personalization and user engagement.

4.2 Disadvantages

Relatively simple

The functions we implement are relatively simple, with only a few basic functions.

Unstable recognition efficiency

Our static Gesture recognition is very accurate, but the dynamic gesture recognition is unstable. The reason is that our implementation logic is based on the change of the vector center of the front and back two frames of images.

5. How to improve

5.1 Provide external API

1. Extensibility

By providing an external interface, I can provide other developers or applications with the ability to access my album's Gesture recognition function. This will facilitate integration and interaction, allowing my features to interact with other applications or devices, thereby expanding their application scope and potential user base.

2. Ecosystem construction:

By opening external interfaces, I can promote a more active and innovative developer community. Other developers can build applications, tools or services based on your Gesture recognition function, so as to enrich the ecosystem of photo album applications and bring more value and creativity.

3. Customization and Personalization:

By improving the interface, I can provide more customization and personalization options, allowing users to adjust the interface layout, theme colors, icons, etc. according to their preferences and needs. This personalized experience can enhance user engagement and satisfaction.

5.2 UI optimization

1. Usability and Ease of Use

Reasonable layout, clear identification and navigation, as well as intuitive operating procedures, can help users quickly grasp functions and operations, and improve the usability and ease of use of the application.

2. Visual appeal

By improving the appearance and visual effects of the interface, I can enhance the visual appeal and quality of the application. A well-designed interface can bring users a pleasant user experience, increase user stickiness and satisfaction.