PROJECT REPORT

Gesture 2048

Team name: orz

Team members: 陈至真、周怡佳、孙雪淳

Teacher: Bao Yang

Table of Contents

Introduction

Our project titled “Gesture 2048” aims to combine the 2048 game with the hand-tracking technique to create a motion sensing game controlled only by hand movements. Our purpose is to not only create a fun game that involves more physical participation, but also explore the feasibility of expanding object tracking’s application to recreational fields.

We were inspired by the demo of object tracking delivered in class. When thinking of a proper game to apply it to, several puzzle games came into our minds including skiing, shooting, Super Mario and so on. But taken into consideration both the compatibility of technique to game and the simplicity of the combination over conventional adaptations, we finally settled on the 2048 game. 2048 is a simple puzzle game which only requires four inputs as four directions, and as a result complements the object tracking function.

The main thread of our project is to implement OpenCV on hand

movement detection and combine it with 2048 game structured on pygame module. To achieve our goals, we consulted GitHub and similar forums for assistance.

Methods & Procedures

A. Preparation: hand detection method

We have to be able to detect hands out of other objects in a video clip first in order to serve the tracking purpose.

This function is realized by a Real-time Hand-Detector using Neural Networks (SSD) on TensorFlow found on GitHub. It uses deep learning frameworks to simplify the process of training a model for object identification and neural networks to support real-time applications of the detection.

B. Getting the coordinates of the detected hands

Once we identify the hands from the picture, we have to generate their coordinates based on which the direction of motion can be acquired.

```
53  def draw_box_on_image(num_hands_detect, score_thresh, scores, boxes, im_width, im_height, image_np):
54      for i in range(num_hands_detect):
55          if (scores[i] > score_thresh):
56              (left, right, top, bottom) = (boxes[i][1] * im_width, boxes[i][3] * im_width,
57                                            boxes[i][0] * im_height, boxes[i][2] * im_height)
58
59              p1 = (int(left), int(top))
60              p2 = (int(right), int(bottom))
61              cv2.rectangle(image_np, p1, p2, (77, 255, 9), 3, 1)
```

Following the pattern of the function which frames the contour of hands given in detector_utils.py, we define a new function to return the location of four sides of the rectangle.

```
67  def record_location(num_hands_detect, score_thresh, scores, boxes, im_width, im_height, image_np):
68      left,right,top,bottom=0,0,0,0
69      for i in range(num_hands_detect):
70          if (scores[i] > score_thresh):
71              left, right, top, bottom = boxes[i][1] * im_width, boxes[i][3] * im_width,boxes[i][0] * im_height, boxes[i][2] * im_height
72          return int(left),int(right),int(top),int(bottom)
```

And then we put the function to the main program and keep the outcomes recorded in a list.

C. Detecting the changes of the coordinates

The next step is to detect the changes of the coordinates to attain the direction of movement. First, we create a list with the length of ten. Then we put in ten coordinate values to compare every two adjacent values. To convert the outcome into one of the four directions, we also calculate the displacement in horizontal direction and vertical direction. This is also compared between every two adjacent values.

```
373         if a > 0 and not location_list[a-1]==[0,0,0,0] and not location_list[a]==[0,0,0,0]:
374         #   水平位移hor和垂直位移ver
375             shift_hor=location_list[a][0]+location_list[a][1] - location_list[a-1][0]-location_list[a-1][1]
376             shift_ver=location_list[a][2]+location_list[a][3] - location_list[a-1][2]-location_list[a-1][3]
377
```

Once the frequencies accumulate to ten, we conduct one count. If there is a biggest number among four directions then this is confirmed to be the direction of the hand. Or else variable hand_direction will have no

value.

```
391             if a==9:
392                 # print(location_list)
393                 # print(num_left,num_right,num_top,num_bottom)
394                 num_max=max(num_left,num_right,num_top,num_bottom)
395                 too_many_max=0
396                 if num_left==num_max:
397                     too_many_max+=1
398                 if num_right==num_max:
399                     too_many_max+=1
400                 if num_top==num_max:
401                     too_many_max+=1
402                 if num_bottom==num_max:
403                     too_many_max+=1
404 
405                 if too_many_max<=1:
406                     if num_left == num_max:
407                         hand_direction = 'left'
408                     elif num_right == num_max:
409                         hand_direction = 'right'
410                     elif num_top == num_max:
411                         hand_direction = 'down'
412                     elif num_bottom == num_max:
413                         hand_direction = 'up'
```

In this way the variable hand_direction can get the real-time moving direction of hands.

D. Changing the input method of 2048 game

```
185     @staticmethod
186     def keyDownPressed(keyvalue, matrix):
187         if keyvalue == K_LEFT:
188             return LeftAction(matrix)
189         elif keyvalue == K_RIGHT:
190             return RightAction(matrix)
191         elif keyvalue == K_UP:
192             return UpAction(matrix)
193         elif keyvalue == K_DOWN:
194             return DownAction(matrix)
195 
196 #修改部分
197     @staticmethod
198     def hand_track_input(keyvalue, matrix):
199         if keyvalue == 'left':
200             return LeftAction(matrix)
201         elif keyvalue == 'right':
202             return RightAction(matrix)
203         elif keyvalue == 'up':
204             return UpAction(matrix)
205         elif keyvalue == 'down':
206             return DownAction(matrix)
```

First, we found a 2048 game with a conventional input method.

Then we define a new function adapted from its original one.

In the meantime, modify the input part of the game.

```
421                     print(hand_direction)
422                     actionObject = GameInit.hand_track_input(hand_direction, matrix)
423                     matrix, score = actionObject.handleData()  # 处理数据
```

E. Combining two modules

We put the two modules under a “while” so that they can open spontaneously. And the input part of 2048 is put under the “if” of hand movement detection. Once the direction is obtained from the ten-element list, it is output to the game.

```
405         if too_many_max<=1:
406             if num_left == num_max:
407                 hand_direction = 'left'
408             elif num_right == num_max:
409                 hand_direction = 'right'
410             elif num_top == num_max:
411                 hand_direction = 'down'
412             elif num_bottom == num_max:
413                 hand_direction = 'up'
414 
415             for event in pygame.event.get():
416                 if event.type == pygame.QUIT:
417                     pygame.quit()
418                     sys.exit(0)
419                 elif event.type == pygame.KEYDOWN:
420                     # actionObject = GameInit.keyDownPressed(event.key, matrix)  # 创建各种动作类的对象
421                     print(hand_direction)
422                     actionObject = GameInit.hand_track_input(hand_direction, matrix)
423                     matrix, score = actionObject.handleData()  # 处理数据
424                     currentscore += score
425                     GameInit.drawSurface(screen, matrix, currentscore)
426                     if matrix.min() != 0:
427                         GameInit.gameOver(matrix)
428                 pygame.display.update()
429                 break
```

At the end of the loop, we reset the value and continue looping.

Problems & Reflection

Problems & Solutions

A. Object identification method

Problem: The current mainstream methods for tracking hands are rule-based such as extracting background based on texture and boundary features, distinguishing between hands and background using colour histograms and HOG classifiers. The drawbacks are that they may easily get confused when the background is unusual or the skin colour has sharp changes.

Solution: The hand-detection using neural networks on GitHub can solve this problem to a great extent. With large datasets, neural networks can train models to perform well in varied conditions. But with deep learning as aid, the process of training a model is simplified and more perfected.

B. Error correction

Problem: Because of the high sensibility of the hand detection technique, small movements will cause errors in direction identification.

Solution: Store ten values in a row and compare in pair. During comparison, we set the default error by 8 so that only shifts larger than 8

will be counted in, and small shakes will be neglected.

```
373         if a > 0 and not location_list[a-1]==[0,0,0,0] and not location_list[a]==[0,0,0,0]:
374         #   水平位移hor和垂直位移ver
375             shift_hor=location_list[a][0]+location_list[a][1] - location_list[a-1][0]-location_list[a-1][1]
376             shift_ver=location_list[a][2]+location_list[a][3] - location_list[a-1][2]-location_list[a-1][3]
377 
378             if abs(shift_hor) > abs(shift_ver):
379                 if shift_hor > 8:
380                     num_left+=1
381                 elif shift_hor < -8:
382                     num_right+=1
383 
384             elif abs(shift_hor) < abs(shift_ver):
385                 if shift_ver > 8:
386                     num_top+=1
387                 elif shift_ver < -8:
388                     num_bottom+=1
```

C. Combination of two modules

Problem: When trying to integrate the game module and the detection module. We faced the difficulty that they can't operate separately at the same time. Directly importing the whole hand-detection program will cause an infinite loop and cannot convey the signal to the game effectively.

Solution: We used a while loop to integrate the codes of two programs in order to spontaneously run both of them, giving real-time feedback to the game.

D. Game Process

```
419                     elif event.type == pygame.KEYDOWN:
```

During the testing we found that after one identification the player must press a random key to start another identification, this might be

caused by:

To press each time may cause some unnecessary operations, but considering we cannot make sure that every movement is what we want to input, especially during the time right after a detection, this is reserved.

Reflection

Our game successfully merges object tracking with a puzzle game, providing players with a more physical and fun way of participating. But we have also encountered many problems during the coding process. The most important issue is how to avoid unwanted inputs. In the case of this project, this can hardly be avoided without interfering the game's fluency and players' experience. The hand can't always move in the wanted direction because when a movement is made the player have to move it back a bit to keep it in coverage. And the detection program cannot distinguish the two kinds of movement, thus causing chaos, harming the game experience. To get rid of this kind of situation, the "random press" procedure must be retained.

And this leads us to think about the feasibility and the future of this kind of games. To actually gain its place in the game market, objecting tracking techniques still have a long way to go. The requirements are not only to recognize the correct target, but also to detect the exact movements made by the target, eliminating interference. After their

perfection, we are positive about its applications in games or even the whole recreational fields.

References

https://github.com/victordibia/handtracking#citing-this-tutorial

https://www.cnblogs.com/darksouls/p/8227821.html