信号与系统大作业之“跳一跳”外挂

漆耘含

无 63

2016011058

一、　外挂模型及算法

1. 基本假设

当玩家按在屏幕上一段时间，小人会向前跳相应的距离。通过数据拟合，可以假设小人下蹲时间 T（即玩家按压时间）与小人跳的距离 L 成正比，系数为 1.392，即 L=1.392T。

通过观察可以发现，不管上一次小人在方块上的位置如何，下一次的落点位置必定在通过下一个块中心的与边线平行的线上，如图 1

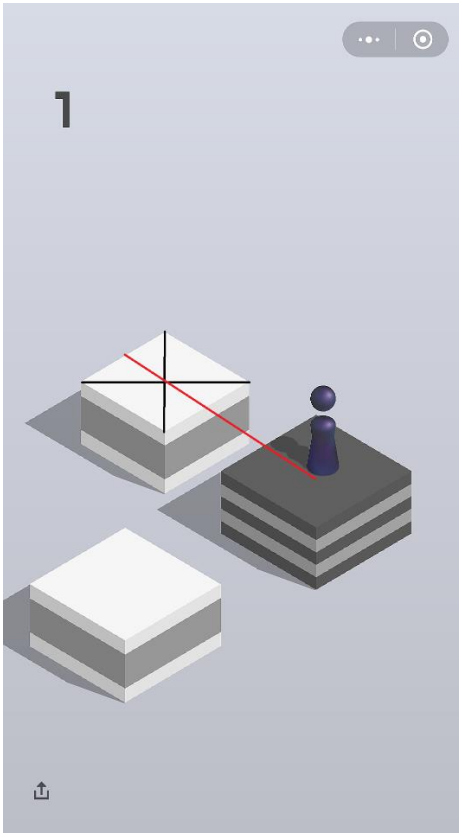

图 1 落点示范图（红线为下一个块的落点）

图像坐标：左上角为坐标原点，竖直向下为 y 轴，水平向右为 x 轴

2. 基本算法

要实现“跳一跳”的外挂程序，需要两个必要的因素：小人的底部中心，下一个块的中心。找小人的底部中心是通过 opencv 的模板匹配功能实现的；找下一个块的中心，

A.　找小人底部的中心点

小人在每一次停住的时候，大小都是不变的，因此这个问题就转变为在截图中寻找小人，并得到其坐标。

在信号与系统的知识中，“**匹配滤波**”是一个很好的选择。匹配滤波是当信号和模板信号匹配（即相等）的时候，会得到一个峰值，而和模板信号不匹配的时候，输出的是一个比较小的值，因此就可以判断出模板所处的位置。

首先，需要得到模板。我先得到一张截图，把它通过一个高斯滤波器把色块变化不均匀的地方模糊化，得到处理后的图像，然后把它从 RGB 转换到灰度图，分别对 x 和 y 方向做梯度，通过 **canny 边缘检测**，得到一张二值图（相当于一个矩阵），这个二值图是 1920*1080 的一个矩阵，里面的元素是 0（黑）和 255（白），可视化结果是一张黑白图，白色的可以明显看出是边缘轮廓，如图 2：

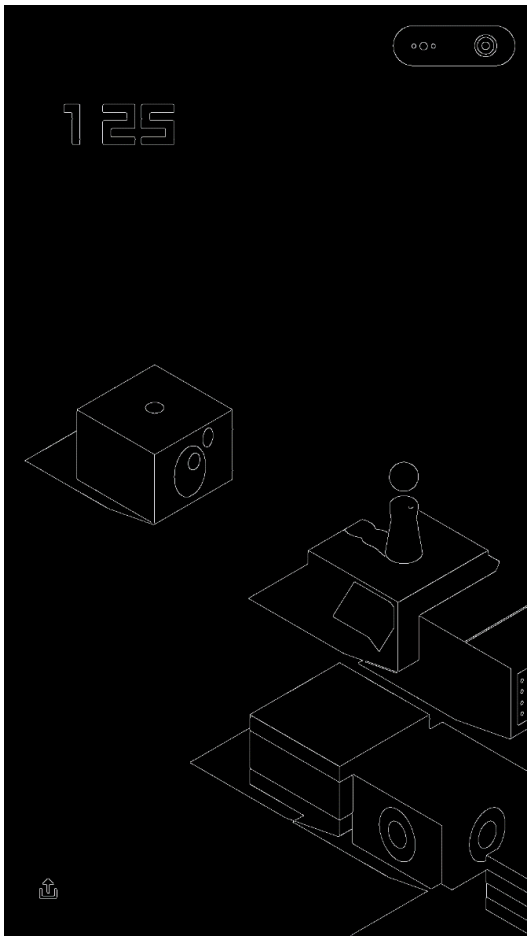

图 2：通过边缘检测的截图示例

小人在刚开始的时候位置是固定不变的，因此可以通过手动把小人所处位置的那一块矩阵挖出来，并用 numpy 的形式保存下来，以供之后进行模板匹配。其可视化图形为图 3：

图 3：小人边缘检测示例

在得到模板之后，需要对每一次的截图进行匹配。也是通过同样的方法，先进行高斯滤波，再进行边缘检测，同样可以得到二值图，然后对模板和目标截图做相关，返回值对大的点（坐标），

即小人的位置，并在小人的底部用一个小圆点标记，如图 4：

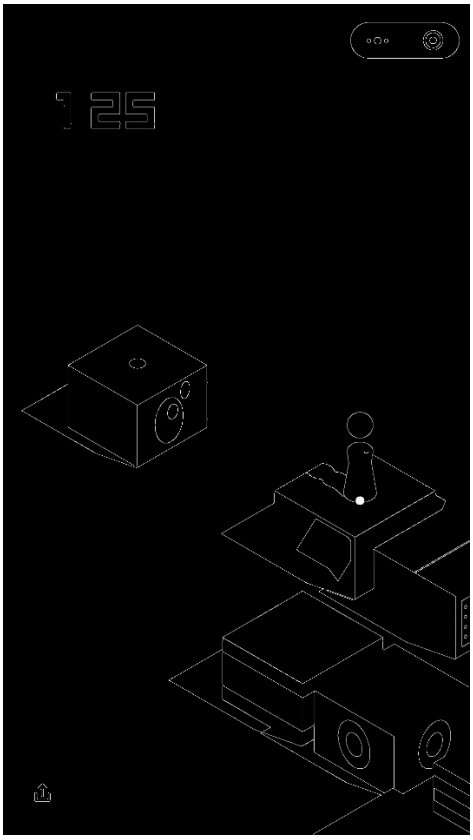

图 4：通过模板匹配之后的小人底部中心

B.　下一个块的中心

找下一个块中心准确与否，直接关系到跳得准不准。我一共想出了三种方法，但最终只选取了效果最好的一种。

第一种是匹配滤波，即通过找出下一个块的最高点（圆弧或三角），根据观察可以得知，小块的上面是对称的，因此可以将最高点附近的图像抠出来，翻折，然后做相关，即可以找到下顶点或下圆弧的位子，两个点连线的中点既是小块的中点。但这种方法有一个短板，就是在某些块的时候，因为其纹理太过复杂，几乎看不出来下顶点，因此匹配的不好，还有就是 python 的矩阵运算速度太慢，因此最后舍弃掉了这种方法，不过这种方法用 matlab 来实现会更好一些。

第二种方法是几何方法，每一次跳的方向是与水平方向呈 30 度，因此可以通过最高点、小人的位置、和方向来确定下一个小块的中心，但这种方法有一个问题，即如果小人的位子偏离了中点的位置，则会连带着影响下一个块中心的检测，误差会逐步累积。如图 5 所示：

第三种是通过截取屏幕中间的一块图像，从上往下、从左往右遍历找到第一个点，同时从上往下、从右往左遍历找到第一个点，这两个点的平均值即为下一小块中点的 x 坐标。然后需要找到最右边的顶点，它的 y 坐标即是中点的 y 坐标。找最右顶点的方法是，从 y 方向上往下遍历，x 方向从右往左遍历，通过遍历可以检测出小块的右边沿，通过记录当前右边沿的 x 坐标，同时还会记录下之前相邻 3 个边沿的 x 坐标，当 y 坐标最小的 x 坐标值大于等于前面三个边沿点的 x 坐标，即可认为该点的 y 坐标是小块中心点的 y 坐标。

通过测试验证，方法一运行速度太慢，方法二误差会偏大，所以都舍弃没用，最后采用的是方法三。

值得注意的是，经过试验，发现截图有时候会把小人的上半部分截进去，因此在进行中点匹配的时候，需要把小人扣掉，具体方法是把小人所在的矩阵值设置为 0（黑）

二、 调试问题

1. 匹配小人底部中点的调试

因为匹配小人底部中点是通过 OpenCV 的模板匹配函数来完成的，返回的最大值是相似度最高的一个坐标，因此通过调用函数得到的点并不是小人的中点，而是需要通过加上特定的常量来调整到相应的值。通过验证，每一次小人匹配返回的坐标和小人的相对位置是一样的，因此在每一张截图中，需要加的常量是定值，不用对每一张截图单独计算需要加的常量。

之前程序返回的 distance 出现问题，是因为我直接把返回的坐标当成小人底部中心的坐标，后来在图中标识出来才发现这个问题。

2. 找小人中点的调试

在写程序的过程中，问题最多的就是检测小块的中心。

上文中提到的方法三，并不是一蹴而就的方法，而是通过不断改进而得来的。最开始找最右点的方法，是从右往左遍历，找到第一个为白色像素点，它的 y 坐标即为中心点的 y 坐标，但是这样效果并不好，要么靠上，要么就靠下，总会有误差。

后来经过分析，发现是因为边缘检测的精确度的问题。因为边缘检测参数设置的问题，导致小块边缘上有一些地方没有连接上，还有一些地方会凸起或者凹陷进去，这个是肉眼很难看出来的，因为可能只是图像矩阵的一小块值。针对这个问题，我把 Canny 函数的值改的比较小，提高了精确性，同时还在进行 Canny 之前，对图像进行了锐化处理，让图片的对比度更高，同时还进行了高斯滤波，防止锐化效果带来的误差影响。

在对图像处理完之后，发现效果好了一些，不过有时也还是有误差，后来仔细思考之后，想出了一种减小误差的方法，即通过连续几个边缘值得 x 坐标大小关系判断，来确定小块的中点坐标，通过这样的改动，最终检测效果非常好。

三、 关键代码

1. Main 函数

```
def main(canny):
    #读取截图
    new_img=Image.open('autojump.png')
    #锐度增强，锐化系数为 1.5
    enh=ImageEnhance.Sharpness(new_img)
    Sharp=1.5
    img_con=enh.enhance(Sharp)
    img_con.save("autojump1.png")#把锐化后的图片保存下来

    #找小人底部中点
    img=cv2.imread('autojump1.png',0)#读取图片
    img=cv2.GaussianBlur(img,(3,3),0)#高斯滤波
    canny_img=cv2.Canny(img,30,150)#进行 canny 边缘检测

    res=cv2.matchTemplate(canny_img,canny,cv2.TM_CCOEFF_
    NORMED)#模板匹配
min_val,max_val,min_loc,max_loc = cv2.minMaxLoc(res)#返回匹配系数最大的坐标和相关系数
    loca=(max_loc[0]+33,max_loc[1]+210)#加上常量到小人底部坐标

    #找下一小块的中点
    canny_img=cv2.Canny(img,5,10)#再次进行边缘检测，提高精度
    #挖掉小人，防止影响
    for y in range(loca[1]-500,loca[1]):
```

```
            for x in range(loca[0]-30,loca[0]+90):
                canny_img[y][x]=0



    c_img=cut(canny_img)#截图
    x_center,y_center=get_center(c_img,loca)#找小块中点，
返回值为中点坐标


    #计算距离
    distance=(loca[0]-x_center)**2+(loca[1]-y_center)**2
    distance=distance**0.5
    return int(distance)
```

2. Getcenter 函数

```
def get_center(canny_img,loca):
    #得到函数的长宽
    crop_h, crop_w = canny_img.shape
    center_x, center_y, center_x1= 0, 0, 0
    flag=0
    flag_y=0
    #从左往右遍历
    for y in range(crop_h):
        for x in range(crop_w):
            if canny_img[y, x] == 255:
                center_x=x
                flag=1
                flag_y=y
                break
```

```
        if flag==1:
            break
    从右往左遍历
    for x in range(crop_w,-1,-1):
        if canny_img[flag_y, x-1] == 255:
            center_x1=x-1
            break
    #中心坐标即为两者的平均值
    center_x=int((center_x1+center_x)/2)
    #寻找最右点
    x_before1=0
    x_before2=0
    x_before3=0
    x_before4=0
    center_y2=0
    flag=0
    x=0
    y=0
    #从上往下，从右往左遍历
    for y in range (crop_h):
        for x in range (crop_w,-1,-1):
            #碰到第一个白色像素点，进行判断操作，完成之后跳出内层循环
            if canny_img[y,x-1]==255:
            center_y2=y;
                #判断的依据
                if ((x_before1 >= x_before2) and (x_before1 >= x-1) and (x_before1 >= x_before3) and (x_before1 >= x_before4)):
```

```
                    flag=1
                else:
                    #进行更新
                    x_before1=x_before2
                    x_before2=x_before3
                    x_before3=x_before4
                    x_before4=x-1
                break;
        if flag == 1:
            break
    #因为中心的 y 坐标是 x_before1，因此要-3
    center_y2 = center_y2-3
    #计算中心的 y 坐标
    center_y=int（center_y2)
    #因为截图是从 300 开始截的，因此在会到原来大截图中，要加上 300
    center_y=center_y+300
    return center_x,center_y
```

3. Cut 函数

```
def cut(canny_img):
    height, width = canny_img.shape
    canny_img = canny_img[300:int(height/2), 0:width]
    return canny_img
```

四、　总结与反思

1. “跳一跳”中心检测问题所用的信号与系统的知识，主要是匹配滤波，但并没有手写匹配滤波，而是通过调用 opencv 的函数来实现的，但是思想是领悟到了，同时也认识到在图像处理中信号与系统的重要地位！
2. 本次使用的是 python 编程，之前并没有了解很多 python，所以很多东西都是这次自学的，从 python 刚入门，到能熟练掌握和应用 python，这是我的一个比较大的收获。
3. 因为时间原因，算法还没有优化到最好，也没有结合更多信号与系统的知识，这是这次大作业的一个缺憾！

五、　参考资料

1. Python 网络教程

   网址：

   http://www.runoob.com/python/python-tutorial.html

2. OpenCV 网络教程:

   https://blog.csdn.net/qq_31531635/article/details/73382603