作业 2：老鼠双目标追踪

一．程序过程概述

程序的输入为视频的名字，输出为每一帧的图片和路径图

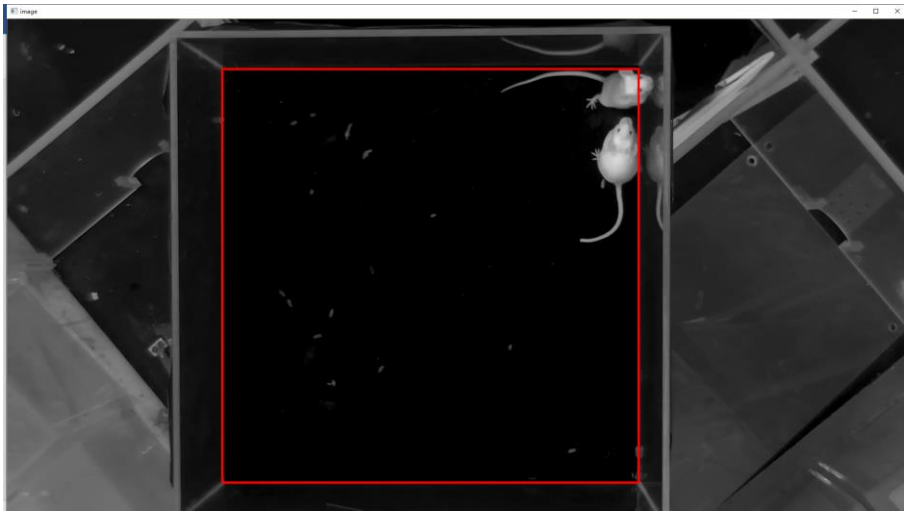
1. 首先，截图每一帧图像；
2. 针对每一帧，把它转化为灰度图；
3. 进行开运算，初步去除噪声；
4. 进行中值滤波，去除椒盐噪声；
5. 进行阈值化处理，85 以下设为 0，85 以上设为 1；
6. 手工选取作业区，进行非选框部分黑化处理来去除部分背景的干扰；
7. Canny 边缘检测，获得边缘图
8. 获取轮廓，画矩形，获得左上角坐标和长宽，即获取了左上角的坐标和右下角坐标；
9. 当检测到只有一个轮廓时，对 6 步得到的灰度图循环进行深度腐蚀，直至得到两个以上轮廓
10. 当检测到多个轮廓时，把相近的合并成一个，只留下两个距离较远的轮廓
11. 进行帧间的匹配，以两张图中同一只老鼠的坐标相近为原则，防止两帧的老鼠的坐标互换。
12. 把本帧的处理后得到的两个坐标保存下来，即更新前帧坐标。
13. 绘图展示，一方面把坐标以圆圈的形式在原图中画出，另一方面创建一个黑背景，通过两点间的连线，来绘制轨迹图。

二．算法具体说明

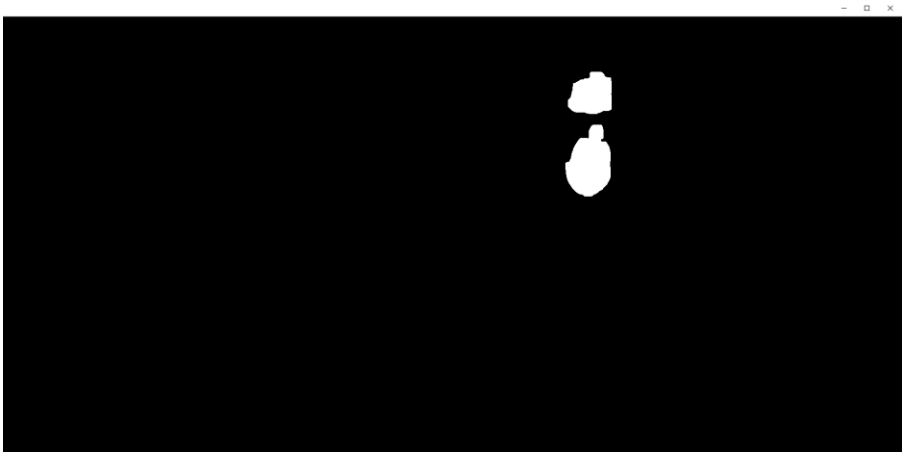
1. 选取作业区
   1) 目的：选定老鼠的活动范围，防止镜子等部分的影响。
   2) 过程：

      首先读取视频的第一帧，通过鼠标左键点击、拖动、松开，由操作者自己选取作业部分。

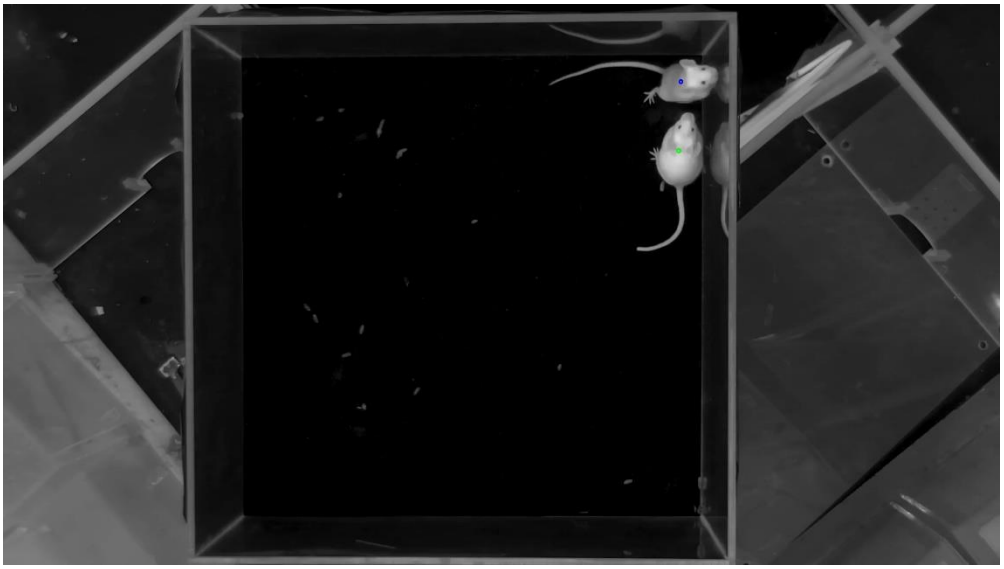
      保存选取选取框（矩形框）的左上角和右下角的坐标，由于相机的除老鼠意外的部分是固定的，所以我们可以直接将第一帧选取的区域作为所有帧的操作区域。
   3) 方法：在这一部分，我们首先建了一个和原视频每张图的大小相等的全 0 矩阵，之后针对每一帧，将选取的框挖出来，再放入全 0 矩阵中，由此得到了一个除了选框内是原图，其他部分全黑的照片。
   4) 效果图

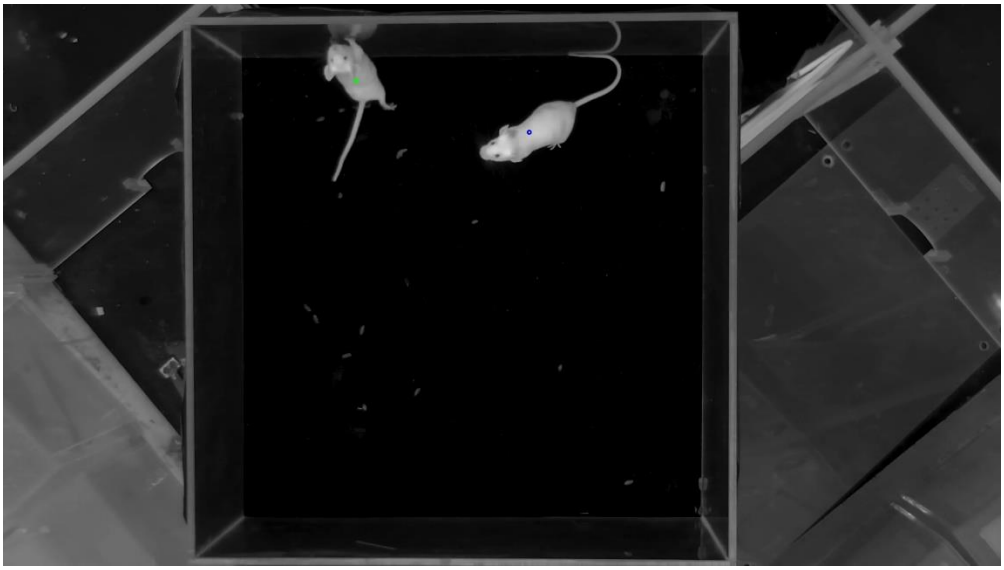


2. 对图片的预处理
   1) 目的：去除噪声和白点区域
   2) 过程：
      截取图片后，转化为灰度图；
      进行掩膜较小的腐蚀处理，减弱白点区域、使老鼠分开、去除老鼠的尾巴；
      进行掩膜较大的膨胀处理，与上一步一起构成开运算，使老鼠的形状变化不会很大，同时去除老鼠内部的空缺，防止之后进行 canny 检测时产生太多边缘；
      进行中值滤波，进一步去噪；
      运用前一步提取非镜子区域。
   3) 效果图

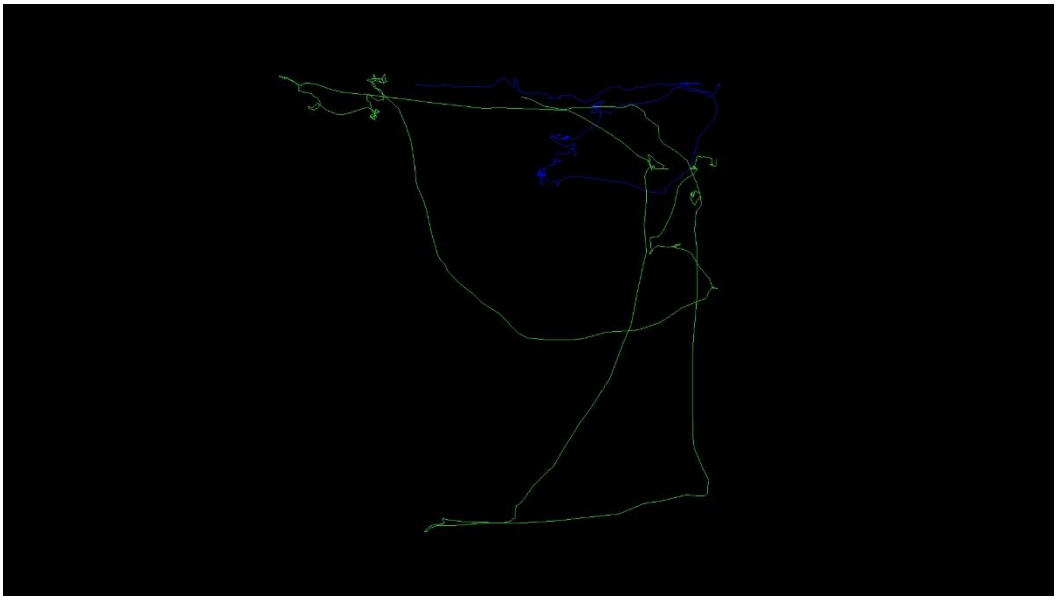


3. 针对某一帧的进一步处理
   1) 目的：获取两个老鼠中心的坐标
   2) 过程：
      首先进行 canny 边缘检测，并获得多个轮廓；
      获取每个轮廓的矩形边框，得到左上角的坐标（x，y）和宽 w，高 h；
      进一步处理得到左上角和右下角的坐标；
      去除 w 和 h 小于 20 的边框，去除小边缘的影响；
      当边框的总数量小于 2（即为 1 时），说明两个鼠没有分开，进行深度腐蚀，返回，对得到的图循环进行上述的操作，直至得到的边框总数大于等于 2；
      当边框总数大于 2 时，说明有的老鼠得到了多余一个边框，但通过分析可知，这一个老鼠的众多边框的中心应该是相近的；
      对所有的边框取中心点，并保存在两组数组 x_n 和 y_n 中，通过循环，当两个坐标点相近（城市距离小于 80）时，去除一个坐标；
      进行完上述操作后，我们便得到了两个小鼠的中心坐标，将其绘制到原图中。
   3) 效果图



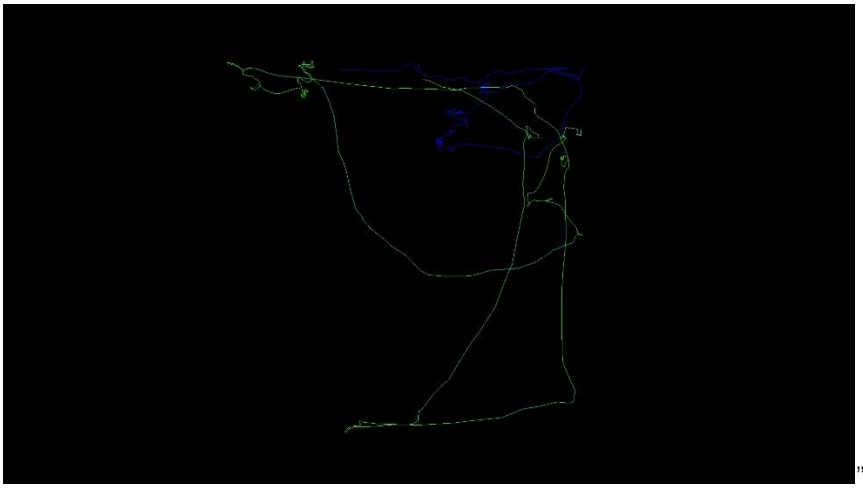


4. 帧间的匹配
   1) 目的：由于每一帧的处理后，只得到了两个坐标，但实际上他们并没有特定的某只老鼠所获得。这一步处理主要是为了保证坐标和老鼠的对应关系。
   2) 过程：
      设置全局标量来保存坐标，初始化为 0；
      把第一帧的坐标保存在全局坐标中；
      在第二帧中，通过最近临原则，把第二帧中临第一帧近的坐标相匹配，并连线，对两只老鼠都进行此操作；
      把本帧的坐标赋值到全局坐标，引入下一帧的处理。
   3) 效果图



三．作业输出

1. 标值老鼠中心位置的视频（图片），见代码运行时的输出
2. 老鼠的轨迹图，见"tracking_mouse.jpg



"

(tracking_mouse.jpg)

3. 源码，见 tracking.py

四．结果分析

1. 算法的适用性：

   算法的背景划分是直接阈值化处理的，算法的原理可以应用于背景和物体对比度明显比较大的场景下，有一定的局限性。

   在应用于不同视频时，膨胀和腐蚀的掩膜的参数和在去除临近的坐标时使用的参数要重新调整。

2. 跟踪准确率：

   在本次追踪中，实现了较完美的追踪，没有出现交叉。

   关于交叉的处理：

   1) 建立全局变量 p1 和 p2 并初始化为 0，用于储存每一帧的两个坐标值（假设 p1 保存第一个小鼠的中心坐标，p2 保存第二个小鼠的中心坐标）；
   2) 对第一帧，将获得的坐标值保存在全局变量中；

3) 对第二帧，将得到的两个坐标分别和 p1 和 p2 作比较，考虑到两只不会同时进行极快的运动，将离 p1 近的点作为第二帧中 p1 鼠对应的坐标，离 p2 近的点作为第二帧中 p2 鼠对应的坐标。之后对两组对应的坐标进行划线；

4) 更新，将第二帧的坐标保存在全局标量中；

5) 引入第三帧，同理进行处理

3. 算法效率：

每秒处理 17 帧

五．源代码

```
import cv2 as cv
import pandas as pd
import numpy as np
import time
import math



#通过与鼠标交互选取操作区域，去除镜子的干扰
global min_r,max_r,min_c,max_c,imgg
#整个图的行列
global rr,cc
#记录帧数
global frame_count
#前帧的坐标值
global x0_f,y0_f,x1_f,y1_f
#按帧截取视频，并进行闭运算和滤波

def capture(video_path):
    #初始化前帧的值
    global x0_f,y0_f,x1_f,y1_f
    global frame_count
    #保存视频的大小
    global rr,cc
    #白板图
    bg=np.zeros((rr,cc,3),dtype=np.uint8)
    #读取视频的预处理
    cap = cv.VideoCapture(video_path)
    cap.isOpened()
    frame_count = 1
    success = True
    kernel1 = np.ones((8,8),np.uint8)#腐蚀运算的掩膜
    kernel2 = np.ones((15,15),np.uint8)#膨胀运算的掩膜
    kernel3 = np.ones((20,20),np.uint8)#深度腐蚀运算的掩膜
    kernel4 = np.ones((5,5),np.uint8)#深度膨胀运算的掩膜
    while (success):
```

```
        success, frame = cap.read()
        if success==0:
            break
        # 转化为灰度图
        frame2=cv.cvtColor(frame,cv.COLOR_BGR2GRAY)
        params = []
        # params.append(cv.CV_IMWRITE_PXM_BINARY) 设置压缩状况
        params.append(1)
        #开运算，去除噪声
        eroded = cv.erode(frame2, kernel1)
        eroded = cv.erode(eroded, kernel1)
        dilated = cv.dilate(eroded, kernel2)
        #中值滤波
        mean_result=cv.medianBlur(dilated, 3)
        #阈值化处理
        r,result1=cv.threshold(mean_result,85,255,cv.THRESH_BINARY)
        #提取非镜子区域
        temp=result1[min_r:max_r,min_c:max_c]
        #将镜子区域涂黑并把非镜子区域放回原图
        r,c=mean_result.shape
        result2=np.zeros((r,c),dtype=np.uint8)
        result2[min_r:max_r,min_c:max_c]=temp
        #canny 边缘检测
        canny_Img = cv.Canny(result2,50,200)
        #获取轮廓
        result3,contours, hierarchy=
cv.findContours(canny_Img,cv.RETR_TREE,cv.CHAIN_APPROX_NONE)
        for i in range(len(contours)):
            sum=0
            x, y, w, h = cv.boundingRect(contours[i])
            if w>20 and h>20:
                sum+=1
       #防止边缘不明显，导致只有一个轮廓
        while sum<2:
            sum=0
            result2 = cv.erode(result2, kernel3)
            canny_Img = cv.Canny(result2,50,200)
            result3,contours, hierarchy=
cv.findContours(canny_Img,cv.RETR_TREE,cv.CHAIN_APPROX_NONE)
            for i in range(len(contours)):
                x, y, w, h = cv.boundingRect(contours[i])
                if w>20 and h>20:
                    sum+=1
```

```
#构建抑制多重临近值的数组
x_n=[]
y_n=[]

#保存所有中心值
for i in range(len(contours)):
    x, y, w, h = cv.boundingRect(contours[i])
    if w>20 and h>20:
        x_n.append(math.floor(x+w/2))
        y_n.append(math.floor(y+h/2))
#去除临近值
if sum>2:
    for i in range(len(x_n)):
        for j in range(i+1,len(x_n)):
            if abs(x_n[i]-x_n[j])+abs(y_n[i]-y_n[j])<80:
                x_n[i]=0
                y_n[i]=0
#去除临近值
while len(x_n)>2:
    x_n.remove(0)
    y_n.remove(0)

#防止两个老鼠的坐标互换,第一帧的处理
if(frame_count==1):
    x0_f=x_n[0]
    y0_f=y_n[0]
    x1_f=x_n[1]
    y1_f=y_n[1]

if(frame_count>=2):
    if abs(x_n[0]-x0_f)+abs(y_n[0]-y0_f)>200:
        temp_x=x_n[0]
        temp_y=y_n[0]
        x_n[0]=x_n[1]
        y_n[0]=y_n[1]
        x_n[1]=temp_x
        y_n[1]=temp_y

#防止两个老鼠的标值互换，第二帧及以后的处理
if(frame_count>1):
    cv.line(bg,(x0_f,y0_f),(x_n[0],y_n[0]),(0,255,0),1)
    cv.line(bg,(x1_f,y1_f),(x_n[1],y_n[1]),(255,0,0),1)
    cv.imshow("tracking_mouse", bg)
    x0_f=x_n[0]
```

```
            y0_f=y_n[0]
            x1_f=x_n[1]
            y1_f=y_n[1]

        #绘图展示
        cv.circle(frame, (x_n[0],y_n[0]), 3, (0,255,0), 2)
        cv.circle(frame, (x_n[1],y_n[1]), 3, (255,0,0), 2)
        cv.imshow("temp",frame)
        cv.waitKey(1)
        # 保存图片
        cv.imwrite("video" + "_%d.jpg" % frame_count, frame, params)
        cv.imwrite("tracking_mouse.jpg",bg)
        frame_count = frame_count + 1
        print(frame_count,'finished')
    cap.release()

#画框的鼠标响应，画框，选择区域
def on_mouse(event, x, y, flags, param):
    global imgg, point1, point2,min_r,min_c,max_r,max_c
    img2 = imgg.copy()
    if event == cv.EVENT_LBUTTONDOWN:         #左键点击
        point1 = (x,y)
        cv.circle(img2, point1, 3, (0,255,0), 2)
        cv.imshow('image', img2)
    elif event == cv.EVENT_MOUSEMOVE and (flags & cv.EVENT_FLAG_LBUTTON):               #按
住左键拖曳
        cv.rectangle(img2, point1, (x,y), (255,0,0), 2)
        cv.imshow('image', img2)
    elif event == cv.EVENT_LBUTTONUP:         #左键释放
        point2 = (x,y)
        cv.rectangle(img2, point1, point2, (0,0,255), 3)
        cv.imshow('image', img2)
        min_c = min(point1[0],point2[0])
        min_r = min(point1[1],point2[1])
        max_c=max(point1[0],point2[0])
        max_r=max(point1[1],point2[1])
#选择，与第画框与选择区域相匹配
def sele(path):
    global imgg,rr,cc
    cap_temp = cv.VideoCapture(path)
    su,fr=cap_temp.read()
    rr=fr.shape[0]
    cc=fr.shape[1]
    imgg=fr
```

```
    cv.namedWindow('image')
    cv.setMouseCallback('image', on_mouse)
    cv.imshow('image', imgg)
    cv.waitKey(0)
    cv.destroyAllWindows()


if __name__=="__main__":
    cv.namedWindow('image')
    path='video_ini.mp4'
    sele(path)
    start_time=time.time()
    capture(path)
    end_time=time.time()
    print("灰度化图片速度：",frame_count/(end_time-start_time),"帧/s")
    cv.destroyAllWindows()
```