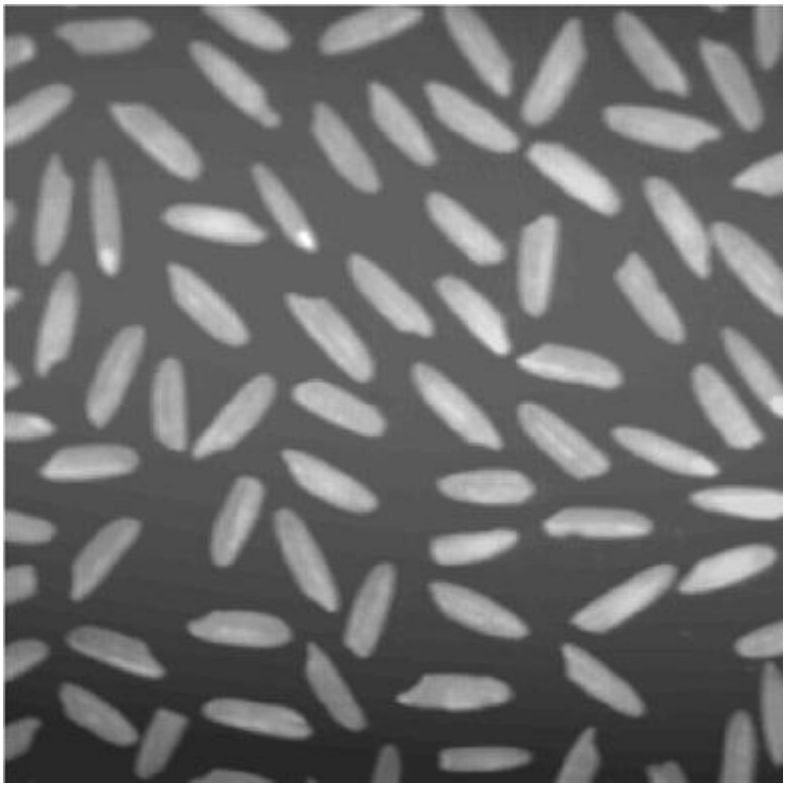
作业一：米粒数量提取



原图

代码如下：

```
import cv2
import numpy as np
img = cv2.imread("rice.jfif")
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) #转换为灰度图

# 使用局部阈值的自适应阈值操作进行图像二值化
dst = cv2.adaptiveThreshold(gray,255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY,101, 1)
# res ,dst = cv2.threshold(gray,0 ,255, cv2.THRESH_OTSU)
# 形态学去噪
element = cv2.getStructuringElement(cv2.MORPH_CROSS,(3, 3))
# 开运算去噪
dst=cv2.morphologyEx(dst,cv2.MORPH_OPEN,element)
# 轮廓检测函数
contours, hierarchy =
cv2.findContours(dst,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
# 绘制轮廓
cv2.drawContours(dst,contours,-1,(120,0,0),2)
```

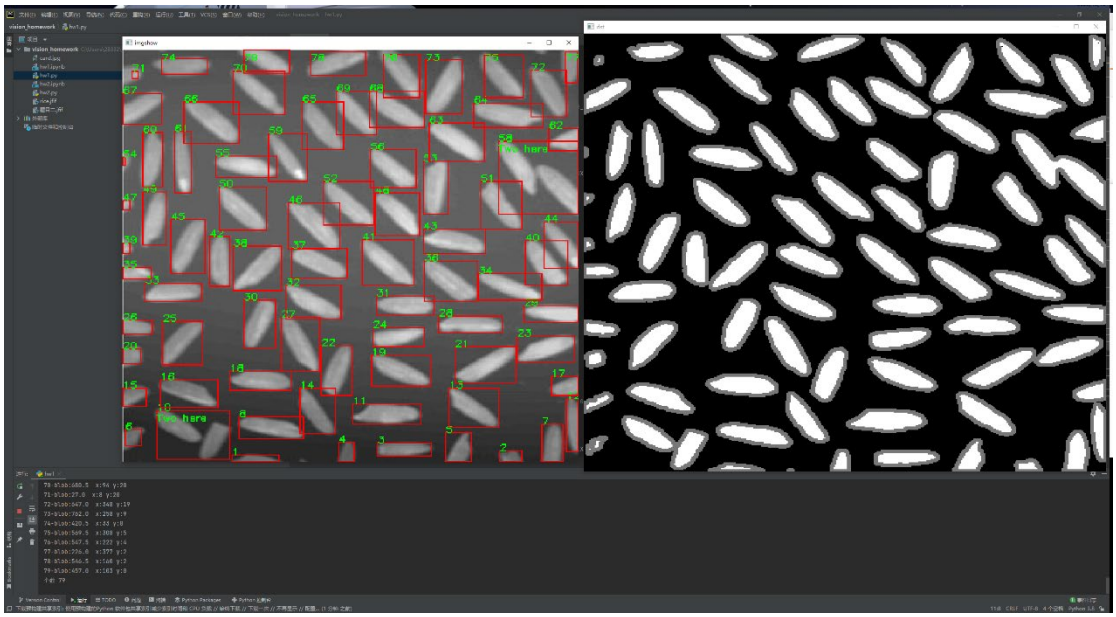
```
count=0 # 米粒总数
# 遍历找到的所有米粒
for cont in contours:
    # 计算包围性状的面积
    ares = cv2.contourArea(cont)
    # 过滤面积小于 50 的形状
    if ares<5:
        continue
    if ares>1000:
        count+=1
    count+=1
    # 打印出每个米粒的面积
    print("{}-blob:{}".format(count,ares),end="  ")
    # 提取矩形坐标（x,y）
    rect = cv2.boundingRect(cont)
    # 打印坐标
    print("x:{} y:{}".format(rect[0],rect[1]))
    # 绘制矩形
    cv2.rectangle(img,rect,(0,0,255),1)
    # 防止编号到图片之外（上面）,因为绘制编号写在左上角，所以让最上面的米粒的 y
小于 10 的变为 10 个像素
    y=10 if rect[1]<10 else rect[1]
    # 在米粒左上角写上编号
    cv2.putText(img,str(count), (rect[0], y), cv2.FONT_HERSHEY_SIMPLEX,
0.3, (0, 255, 0), 1)
    if ares>1000:
        cv2.putText(img,"Two here", (rect[0], y+10),
cv2.FONT_HERSHEY_SIMPLEX, 0.3, (0, 255, 0), 1)
    # print('编号坐标: ',rect[0],' ', y)
print('个数',count)

cv2.namedWindow("imgshow", cv2.WINDOW_NORMAL)   #创建一个窗口
cv2.imshow('imgshow', img)    #显示原始图片（添加了外接矩形）

cv2.namedWindow("dst", cv2.WINDOW_NORMAL)   #创建一个窗口
cv2.imshow("dst", dst)  #显示灰度图

cv2.waitKey()
```

得到的结果：米粒数量为 79



作业二：仿射变换校正畸变

原图

代码如下：

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def Cal_kb_linear_fitline(data_line1):

    output = cv2.fitLine(data_line1, cv2.DIST_L2, 0, 0.01, 0.01)

    k = output[1] / output[0]
    b = output[3] - k * output[2]

    return k,b
img = cv2.imread("card.jpg")

gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) #转换为灰度图
blur = cv2.GaussianBlur(gray, (5, 5), 0)
dst1 = cv2.Canny(blur, 75, 200)
```

```
contours, hierarchy =
cv2.findContours(dst1,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
# 绘制所有轮廓
cv2.drawContours(dst1,contours,-1,(120,0,0),2)
plt.subplot(221), plt.imshow(cv2.cvtColor(dst1, cv2.COLOR_BGR2RGB)),
plt.title('dst1')

areas = []

for c in range(len(contours)):
    areas.append(contours[c].shape[0])

max_id = areas.index(max(areas))

maxContour = contours[max_id]
# 绘制最大的轮廓（即卡片的边框轮廓）
dst2 = cv2.drawContours(img,maxContour,-1,(0,255,0),2)
plt.subplot(222), plt.imshow(cv2.cvtColor(dst2, cv2.COLOR_BGR2RGB)),
plt.title('dst2')

# 将图片划分为 25x25 个区域，分别对其中的点进行拟合，算出所有分区中的斜率和截距
yMag = img.shape[0]
xMag = img.shape[1]
print(img.shape)
lines = []
for i in range(25): # x
    for j in range(25): # y
        points = []
        for r in range(maxContour[:, 0, :].shape[0]):
            point_y = maxContour[r, 0, 1]
            point_x = maxContour[r, 0, 0]
            if (point_x >= i*xMag/25 and point_x < (i+1)*xMag/25) and
(point_y >= j*yMag/25 and point_y < (j+1)*yMag/25):
                points.append([point_x, point_y])
        npPoints = np.array(points)
        if npPoints.size != 0:
            output = cv2.fitLine(npPoints, cv2.DIST_L2, 0, 0.01, 0.01)
            k = output[1] / output[0]
            b = output[3] - k * output[2]
            lines.append(np.array([k, b]))

# 将所得的斜率和截距进行比较，取得中间出现最多的四组解，即为卡片边框的四条边的
解
lines = np.array(lines)
```

```
names = locals()
i = 0
while(lines.size != 0):
    absK = abs(lines[:, 0, 0] - lines[0, 0, 0])
    absB = abs(lines[:, 1, 0] - lines[0, 1, 0])
    idx = np.where(np.logical_and(absK < 0.5, absB < 150))
    names['KBs' + str(i)] = np.vstack((lines[:, 0, 0][idx], lines[:, 1,
0][idx]))
    lines = np.delete(lines, idx, 0)
    i += 1
t = []
for j in range(i):
    t.append(names['KBs' + str(j)].size)

max_number = []
max_index = []
for _ in range(4):
    number = max(t)
    index = t.index(number)
    t[index] = 0
    max_number.append(number)
    max_index.append(index)
Ks = []
Bs = []
for idx in max_index:
    Ks.append(np.average(names['KBs' + str(idx)][0, :]))
    Bs.append(np.average(names['KBs' + str(idx)][1, :]))

def cross_point(k1, b1, k2, b2):  # 计算交点函数
    #是否存在交点
    point_is_exist=False
    x = (b2 - b1) * 1.0 / (k1 - k2)
    y = k1 * x * 1.0 + b1 * 1.0
    if (x > 0 and x < xMag) and (y > 0 and y < yMag):
        point_is_exist=True
    return point_is_exist,[x, y]

# 由四条边算出原图中卡片的四个角点（这里也计算了落在屏幕外的点的位置，通过图片
大小为阈值而舍弃）
pts1 = []
for i in range(4):
    for j in range(i, 4):
        point_is_exist,[x, y] = cross_point(Ks[i], Bs[i], Ks[j], Bs[j])
        if(point_is_exist == True):
```

```
            pts1.append([x, y])

pts1 = np.array(pts1).astype(int)
for i in range(4):
    ptStart = (0, int(Bs[i]))
    ptEnd = (4000, int(4000*Ks[i]+Bs[i]))
    dst3 = cv2.line(img, ptStart, ptEnd, (0, 0, 255), 2)
    cv2.circle(dst3, pts1[i], 10, (0, 0, 255), 5)

plt.subplot(223), plt.imshow(cv2.cvtColor(dst3, cv2.COLOR_BGR2RGB)),
plt.title('dst3')

pts1 = np.float32(pts1)
pts2 = np.float32([[0, 0], [1000, 0], [0, 750], [1000, 750]])
# 生成透视变换矩阵
M = cv2.getPerspectiveTransform(pts1, pts2)
# 进行透视变换
dst4 = cv2.warpPerspective(img, M, (1000, 750))

plt.subplot(224), plt.imshow(cv2.cvtColor(dst4, cv2.COLOR_BGR2RGB)),
plt.title('dst4')
plt.show()
```

结果如下: