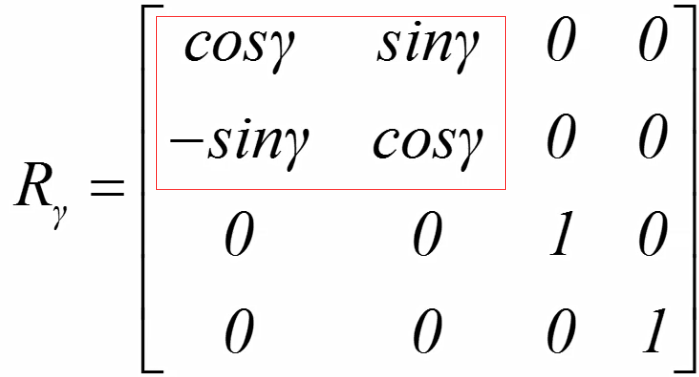
图像旋转

**一 实验原理**

本次代码分为两个部分和一个附加部分。

**第一部分：**以2xN矩阵形式给出一组点，给出要求的旋转角度，返回一个包含旋转之后点的矩阵。使用矩阵相乘获得旋转之后的点，图一是在围绕z轴时的旋转矩阵，因为在本次处理中只需要处理二维的数据，所以只取矩阵左上角红色方框包含的部分。

因为该矩阵给出的是顺时针旋转的情况，且python的三角函数计算需要输入弧度而我们使用的是角度，所以需要做两步处理。第一步，将输入的角度除以180°乘获得弧度。第二步，修正旋转矩阵或处理输入角度，修正矩阵等于进行了一次或的变换。



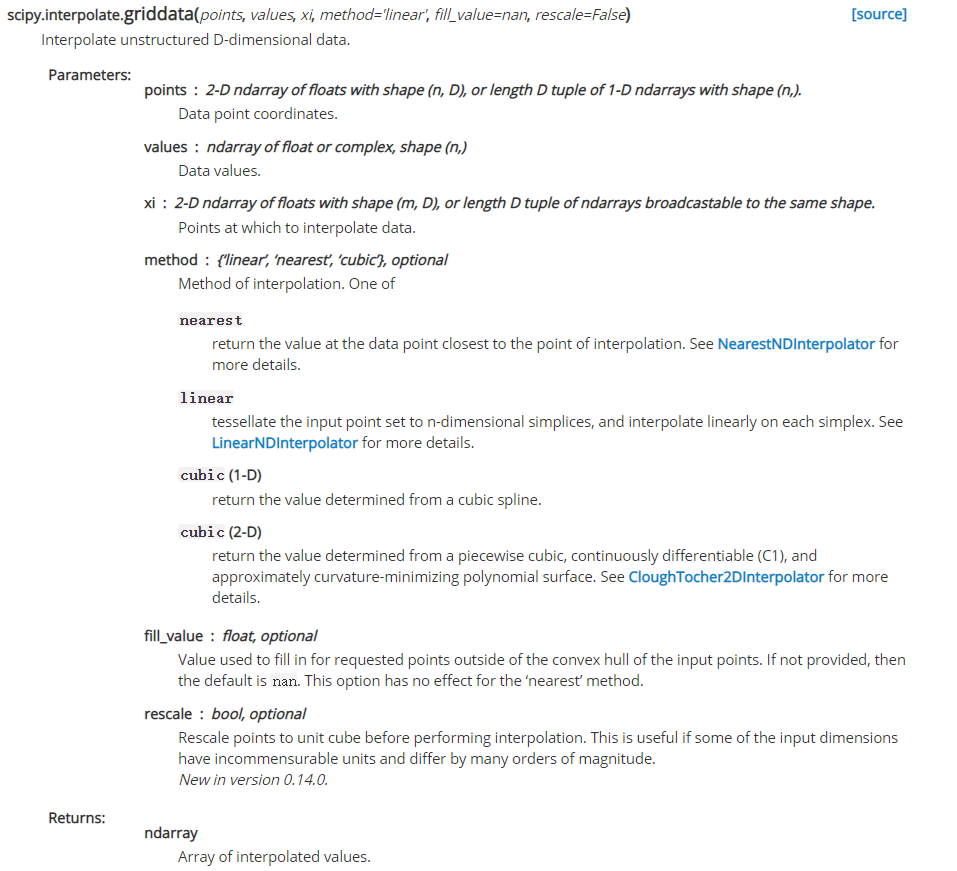
图一：旋转矩阵

**第二部分：**利用第一部分的结果，完成对图像的旋转。第二部分大致分为两步工作，第一步为准备griddata函数所需要的数据，第二步为正确调用griddata函数并创建图像。

由于图像的坐标以左上角为起始点，且排列方向与普通笛卡尔坐标系不同，所以需要对原图像的点进行坐标变换。变换规则是将当前点下标与中心点下标比较，计算获得以中心点为原点的二维坐标系下各个点的坐。由于二维矩阵无法很好的计算旋转，在这里我先创建一个存储原图像所有点的坐标的2xN矩阵，然后设计下标的映射规则，这里设置为2xN矩阵的第二维下标。

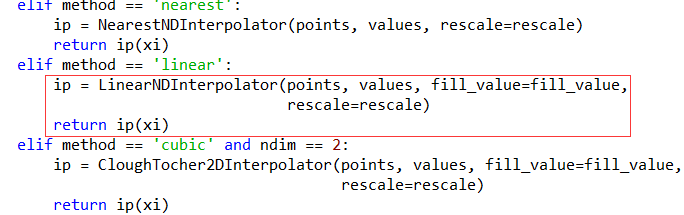
在完成坐标变换后进行旋转变换，调用第一部分的程序，将2xN矩阵中的坐标变换为旋转后的坐标。应注意的是，此时矩阵中存储的依旧是原图像各个点相对于原点新坐标，应将这部分点映射到新的二维图像中。遍历2xN矩阵获得新图像的边界。

接着是griddata函数的理解和调用。在文档中给出了matlab下该函数的使用方法，幸而在python的scipy库中同样有这个函数。Griddata需要至少三个参数才能够工作。这里我参考了scipy的官方文档[1]，见图二。

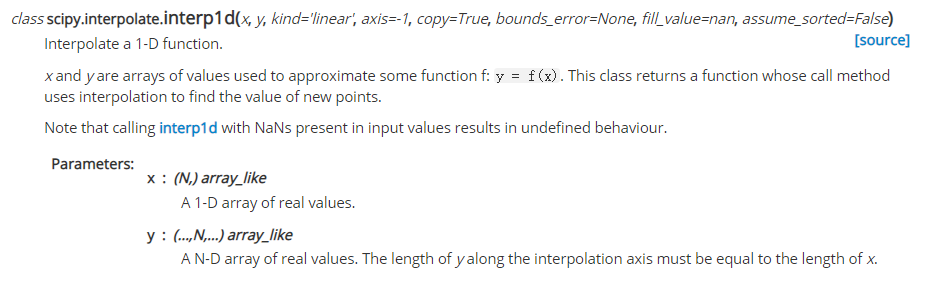


图二：scipy官方文档

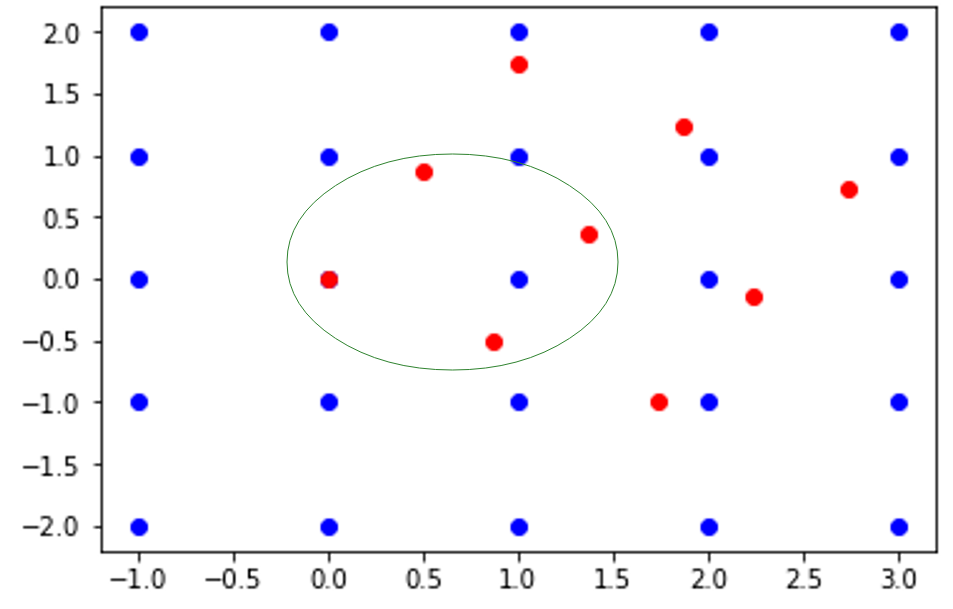
在本次实验中，第一个参数points输入之前计算好的2xN矩阵，需要注意的是这里矩阵需要进行一次转置。第二个参数values是每个点的值，这里我设置一个Nx3数组，对于原图像( i , j )处的点的像素以的规则存储。第三个参数xi是网格点，也就是在新的坐标系中每个网格点的坐标值。这里我看了一下griddata的源码如图三，在红色方框内调用的函数来自scipy的库interpolate.interpnd，这个库由于win10权限没能看到源码，不过我找到了scipy的说明文档[2]，说明见图四，可以看到是在对一个新的坐标视图进行操作，返回新的各个网格点的值。显然这之中的插值方法与第一次作业密切相关，见图五，蓝色的点是网格点，红色的点是有像素值的点，在绿色椭圆形圈内可以看到有四个有像素的点和一个网格点，网格点通过线性插值获得像素。



图三：griddata源码



图四：interpnd说明

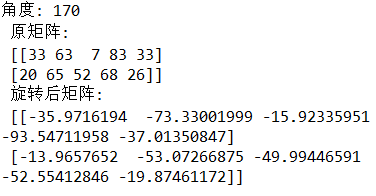
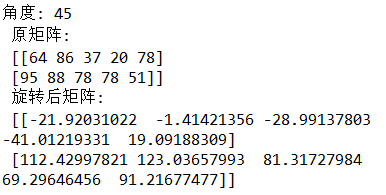


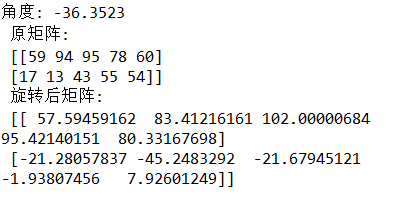
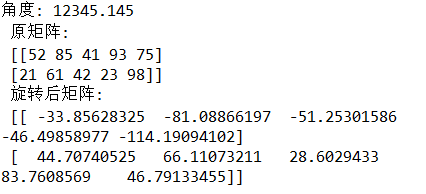
图五：网格说明

**第三部分：**创建一个交互界面，操作者在图像上点击两次获得一条连线，旋转图像，使这条连线水平。我使用了opencv所带的鼠标响应函数库[3]，cv的官方文档写的不是很详细，参考了其他一些说明[4,5]。不断监听鼠标事件，存储当前点击的鼠标坐标和上一次点击的鼠标坐标，当鼠标点击次数大于零且是偶数时旋转一次图片。因为在多次旋转中，需要记录这次旋转和上次旋转的相对旋转角度，所以设置总体旋转角度，每次旋转时叠加到总体旋转角度并显示原图片经总体旋转角度旋转后的图片。

**二 结果展示**

**第一部分：**





图六：原矩阵和旋转后矩阵

**第二部分：**

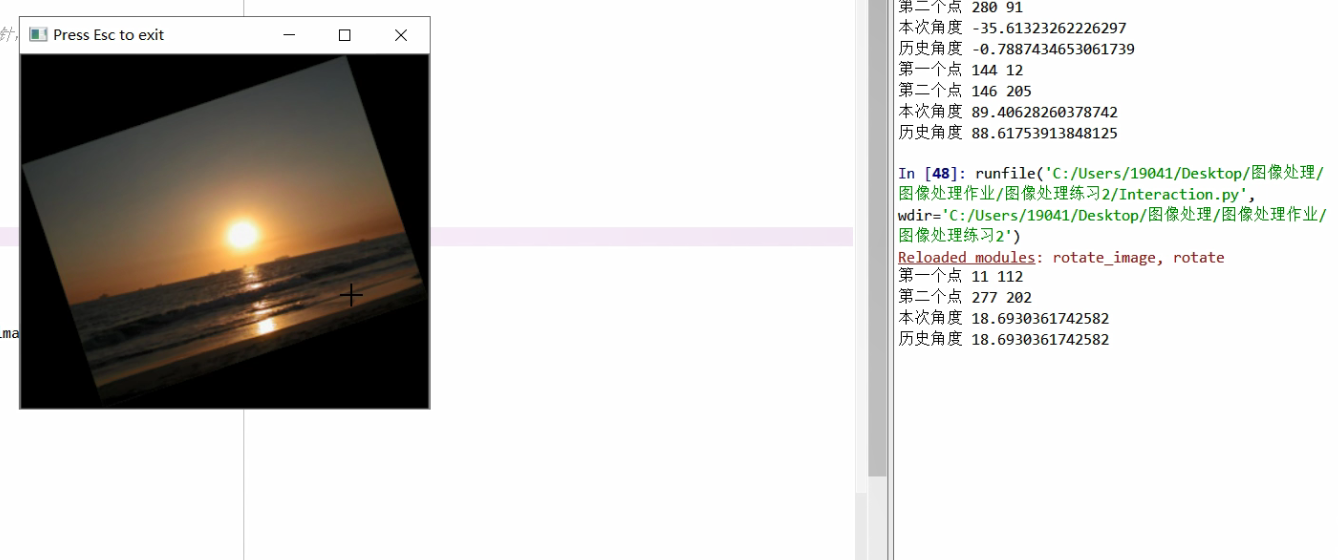


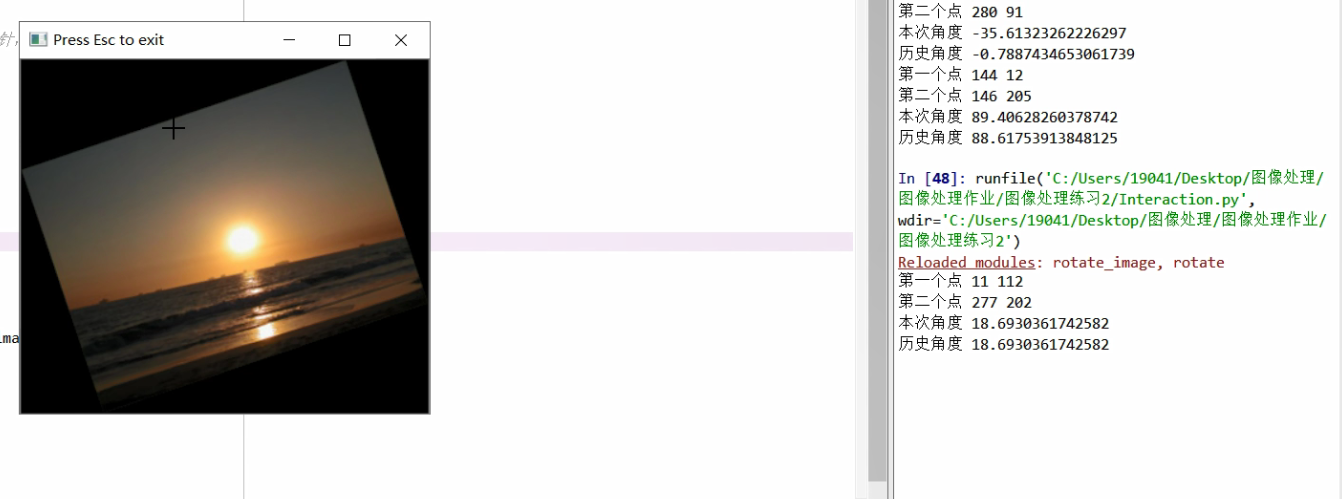
图八：在文件系统中显示的原图与旋转后的图像

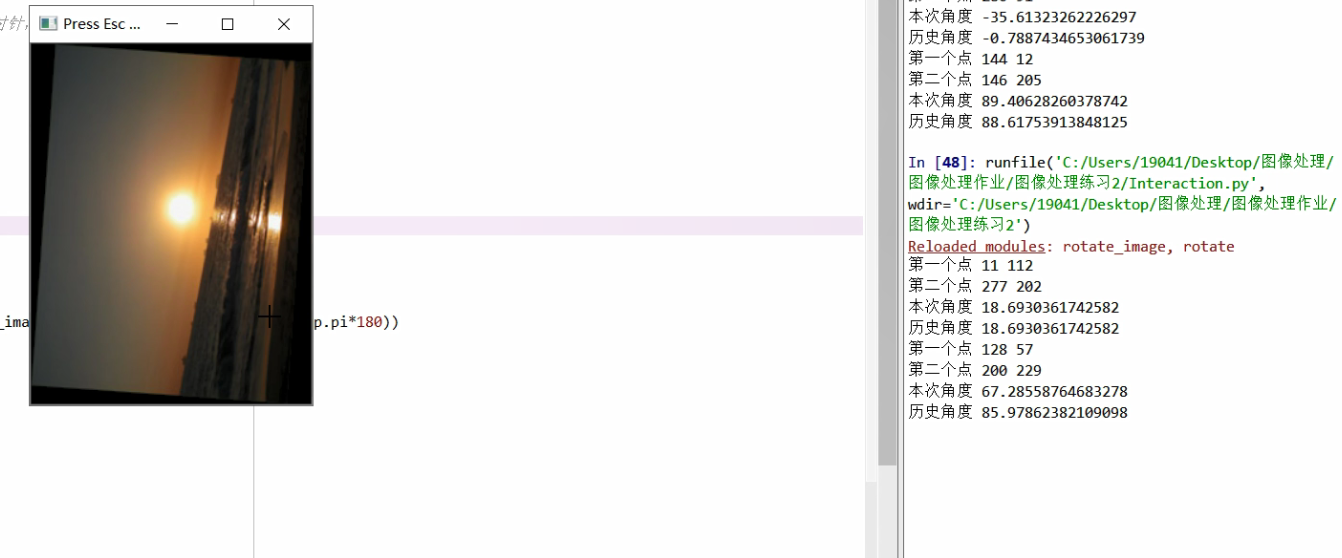


图七：原图与旋转15°后的图像

**第三部分：**

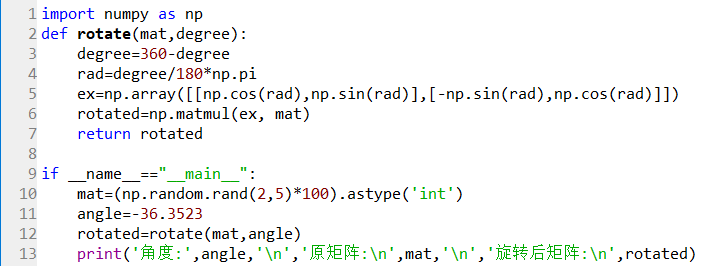




图九：操作关键步骤截图

**三 代码**

**第一部分代码：**

****

import numpy as np

def rotate(mat,degree):

degree=360-degree

rad=degree/180\*np.pi

ex=np.array([[np.cos(rad),np.sin(rad)],[-np.sin(rad),np.cos(rad)]])

rotated=np.matmul(ex, mat)

return rotated

if \_\_name\_\_=="\_\_main\_\_":

mat=(np.random.rand(2,5)\*100).astype('int')

angle=-36.3523

rotated=rotate(mat,angle)

print('角度:',angle,'\n','原矩阵:\n',mat,'\n','旋转后矩阵:\n',rotated)

**第二部分代码：**





import numpy as np

import imageio

import rotate

import scipy.interpolate

import matplotlib.pyplot as plt

#https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html

def rotate\_image(im,angle):

h=im.shape[0]

w=im.shape[1]

midx=(w-1)/2

midy=(h-1)/2

max\_h=-np.Infinity

max\_w=-np.Infinity

total\_point=h\*w

t\_xy=np.zeros((2,total\_point))

values=np.zeros((w\*h,3))

#获取t\_yx和values

for i in range(im.shape[0]):

for j in range(im.shape[1]):

t\_xy[0][i\*w +j]=j-midx#i,j与x，y坐标的换算

t\_xy[1][i\*w +j]=midy-i

values[i\*w+j]=im[i][j]

#np.arctan(h/w)/np.pi\*180 测试左上角点是否旋转至水平

t\_xy=rotate.rotate(t\_xy,angle)

#获取边界

for i in range(len(t\_xy[0])):

if t\_xy[0][i]>max\_w:

max\_w=t\_xy[0][i]

if t\_xy[1][i]>max\_h:

max\_h=t\_xy[1][i]

#矩阵转置

t\_xy=np.transpose(t\_xy)

#新图像大小

new\_h=int(max\_h\*2+1)

new\_w=int(max\_w\*2+1)

new\_im=np.zeros((new\_h,new\_w,3))

new\_h=new\_im.shape[0]

new\_w=new\_im.shape[1]

new\_midx=(new\_w-1)/2

new\_midy=(new\_h-1)/2

#获取网格点

xi=np.zeros((new\_h\*new\_w,2))

for i in range(new\_im.shape[0]):

for j in range(new\_im.shape[1]):

xi[i\*new\_w +j][0]=j-new\_midx#i,j与x，y坐标的换算

xi[i\*new\_w +j][1]=new\_midy-i

#使用scipy函数griddata实现，t\_xy代表新的点的坐标，values代表点的值，xi是网格点的坐标

new\_im\_values=scipy.interpolate.griddata(t\_xy, values, xi)

for i in range(new\_im.shape[0]):

for j in range(new\_im.shape[1]):

new\_im[i][j]=new\_im\_values[i\*new\_w+j]

return new\_im.astype('uint8')

def show(im):

plt.ion()

plt.imshow(im)

plt.show()

def main(path):

im=imageio.imread(path)

show(im)

angle=1414141

im=rotate\_image(im,angle)

str\_list = list(path)

str\_list.insert(-4,str(angle))

tpath = ''.join(str\_list)

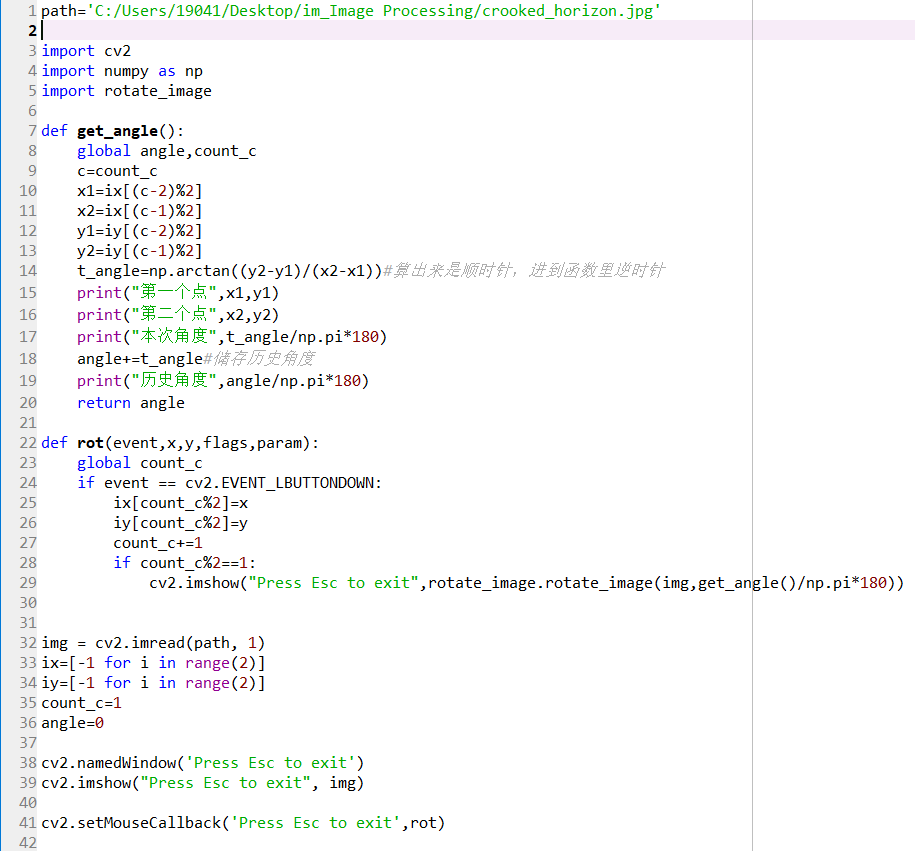
imageio.imwrite(tpath,im)

show(im)

if \_\_name\_\_ == '\_\_main\_\_':

main('C:/Users/19041/Desktop/im\_Image Processing/crooked\_horizon.jpg')

**第三部分代码：**



path='C:/Users/19041/Desktop/im\_Image Processing/crooked\_horizon.jpg'

import cv2

import numpy as np

import rotate\_image

def get\_angle():

global angle,count\_c

c=count\_c

x1=ix[(c-2)%2]

x2=ix[(c-1)%2]

y1=iy[(c-2)%2]

y2=iy[(c-1)%2]

t\_angle=np.arctan((y2-y1)/(x2-x1))#算出来是顺时针，进到函数里逆时针

print("第一个点",x1,y1)

print("第二个点",x2,y2)

print("本次角度",t\_angle/np.pi\*180)

angle+=t\_angle#储存历史角度

print("历史角度",angle/np.pi\*180)

return angle

def rot(event,x,y,flags,param):

global count\_c

if event == cv2.EVENT\_LBUTTONDOWN:

ix[count\_c%2]=x

iy[count\_c%2]=y

count\_c+=1

if count\_c%2==1:

cv2.imshow("Press Esc to exit",rotate\_image.rotate\_image(img,get\_angle()/np.pi\*180))

img = cv2.imread(path, 1)

ix=[-1 for i in range(2)]

iy=[-1 for i in range(2)]

count\_c=1

angle=0

cv2.namedWindow('Press Esc to exit')

cv2.imshow("Press Esc to exit", img)

cv2.setMouseCallback('Press Esc to exit',rot)

**四 优化与griddata的另一种方式**

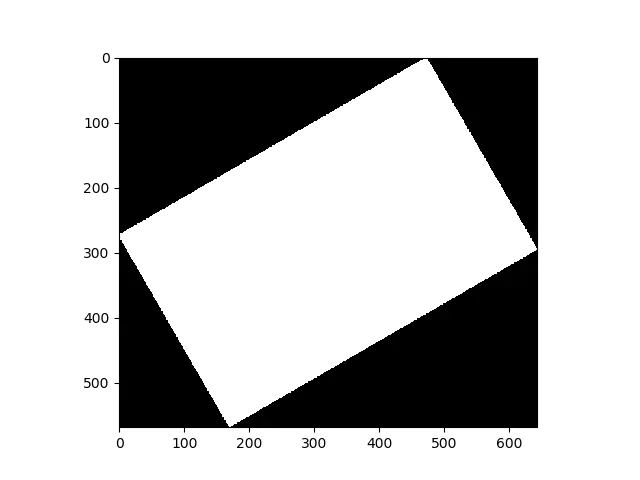
**优化：**本次实验待优化之处有二

其一运算速度较慢，当使用Numpy 数组进行基本的数组计算时，Numpy 都会自动在内部对数组进行向量化，这也是它可以比原生Python代码有更好性能的原因[6]。但是由于我的代码中有大量的循环，在循环遍历一个numpy数组时与原生python速度无提升，甚至因为numpy需对数组操作进行类型检测，可能会出现比原生python更慢的情况。这个时候可以进行的优化方式有二，一是使用numba、cython、pypy等jit来加速，二是尽可能的减少循环，将循环代码的逻辑用numpy的矩阵乘法、数组乘法、数组加法等代替。

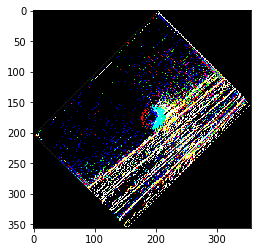
其二是在写第三部分程序时，受限于cv的鼠标调用机制，不得不设置了数个全局变量，为代码的引用带来了不便。这里的解决思路是使用面向对象规范重写代码，将全局变量的调用改为类的内部变量调用，提升内聚性、降低功能间耦合。

**另一种griddata：**逆运算与后向映射

在本次实验中使用的旋转方法是根据当前图像计算出旋转后的像素坐标，然后根据浮点数格式的像素坐标对网格点进行插值。但我在查阅资料的过程中发现了不同的旋转方法[7]，即先假设有一张空白的目标图片如图十[8]，其中白色部分是原图片旋转之后所占据的部分。将原图片的旋转看作目标图片的逆向旋转，这样目标图片的像素坐标是整数，目标图片旋转到原图片的像素对应坐标为实数，使用原图像的网格点对目标图片的旋转进行插值。图十一是我是用网络上提供的逆向旋转方法得到的图片与我的图片的差。



图十：逆向旋转



图十一：正逆向旋转图片之差

**五 参考与引用**

[1] <https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html>

[2] <https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html>

[3] <https://docs.opencv.org/master/db/d5b/tutorial_py_mouse_handling.html>

[4] <https://www.cnblogs.com/mtcnn/p/9412000.html>

[5] <https://blog.csdn.net/akadiao/article/details/80312254>

[6] <https://blog.csdn.net/Pythoncxy/article/details/93040601>

[7] <https://blog.csdn.net/liyuan02/article/details/6750828>

[8] <https://www.jianshu.com/p/04b7e7c36cfa>