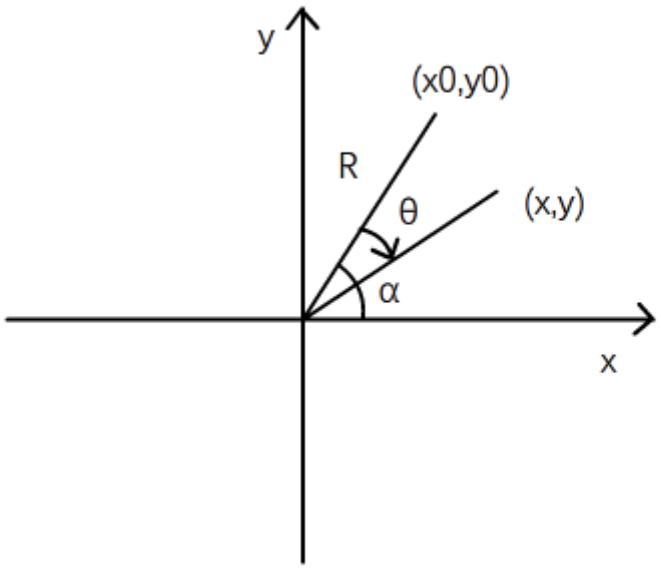
图像旋转是图像处理中最常见的操作之一，本文主要介绍图片旋转的原理，以及用 Python 实现图片的旋转。

1. 旋转矩阵



如上图所示，点(x0,y0)旋转到点(x,y)处，并假设该点离原点的距离为 R。容易可知有以下公式：

$$x = R \cdot cos(\alpha - \theta)$$
$$y = R \cdot sin(\alpha - \theta)$$

化简，可得：

$$\begin{aligned} x &= R \cdot cos(\alpha - \theta) \\ &= R \cdot cos\alpha \cdot cos\theta + R \cdot sin\alpha \cdot sin\theta \\ &= x_0 cos\theta + y_0 sin\theta \end{aligned}$$

$$\begin{aligned} y &= R \cdot sin(\alpha - \theta) \\ &= R \cdot sin\alpha \cdot cos\theta - R \cdot cos\alpha \cdot sin\theta \\ &= y_0 cos\theta - x_0 sin\theta \end{aligned}$$

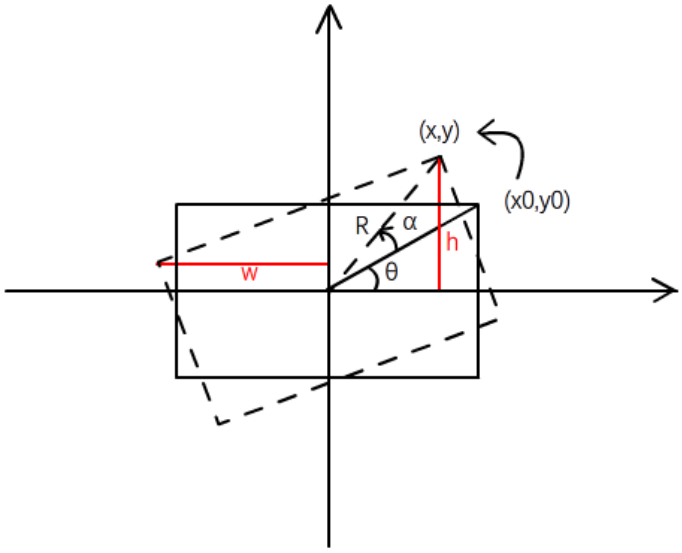
使用矩阵表示：

$$[x \quad y \quad 1] = [x_0 \quad y_0 \quad 1] \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$[x_0 \quad y_0 \quad 1] = [x \quad y \quad 1] \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. 计算旋转后的图像的宽与高

与计算旋转点坐标的方法类似，通过画图可以很容易算出旋转后的图像的宽和高。



$$
\begin{aligned}
\frac{h}{2} &= R \cdot \sin(\alpha + \theta) \\
&= R \cdot (sin\alpha \cdot cos\theta + cos\alpha \cdot sin\theta) \\
&= \frac{W}{2} \cdot sin\alpha + \frac{H}{2} \cdot cos\alpha \\
h &= W \cdot sin\alpha + H \cdot cos\alpha
\end{aligned}
$$

相似地，可得：

$$
2w = W \cdot cos\alpha + H \cdot sin\alpha
$$

3. 数学坐标系与图像坐标系的转换

图像的旋转是以图像的中心点为参考点，对应着数学坐标系的原点，而图像坐标系是以图像左上角为原点。因此我们需要在图像旋转时实现数字坐标与图像坐标的相互转换。

假设原图像的大小为 W，H，旋转后的图像的最小矩形大小为 W'，H'，则从图像坐标系点(x0,y0)到对应的数字坐标点(x,y)有以下转换公式：

$$
[x \quad y \quad 1] = [x_0 \quad y_0 \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -0.5W & 0.5H & 1 \end{bmatrix}
$$

$$
[x_0 \quad y_0 \quad 1] = [x \quad y \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5W' & 0.5H' & 1 \end{bmatrix}
$$

结合旋转矩阵，我们得到最终的旋转变换公式：

$$
[x \quad y \quad 1] = [x_0 \quad y_0 \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -0.5W & 0.5H & 1 \end{bmatrix} \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5W' & 0.5H' & 1 \end{bmatrix}, \quad ①
$$

$$[x_0 \quad y_0 \quad 1] = [x \quad y \quad 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5W' & 0.5H' & 1 \end{bmatrix}\begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5W' & 0.5H' & 1 \end{bmatrix}, \quad ②$$

上面有两条变换公式，实际上式①为前向映射变换，式②为后向映射变换。

4. 前向映射与后向映射

通俗的讲，前向映射就是遍历原图的点，将其映射到旋转后的图像上。后向映射，就是遍历旋转后的图像上的点，然后在原图上找到对应的像素点，将其填入。

一般我们选择后向映射，因为前向映射得到的坐标是浮点型，而坐标只能是整型，会导致旋转后的图像某些像素缺失。采用后向映射则可以保证图像每个像素点都是有值的。

后向映射主要需要解决插值问题，一般会有最近邻法(Nearest Interpolation)、双线性插值(Bilinear Interpolation)、双三次插值(Bicubic interpolation)。

4.1 最近邻法

最邻近算法不需要计算新图像矩阵中点的数值，而是直接找到原图像中对应的点，将数值赋值给新图像矩阵中的点。根据对应关系找到原图像中的对应的坐标，这个坐标可能不是整数，这时候找到最近的点进行插值。

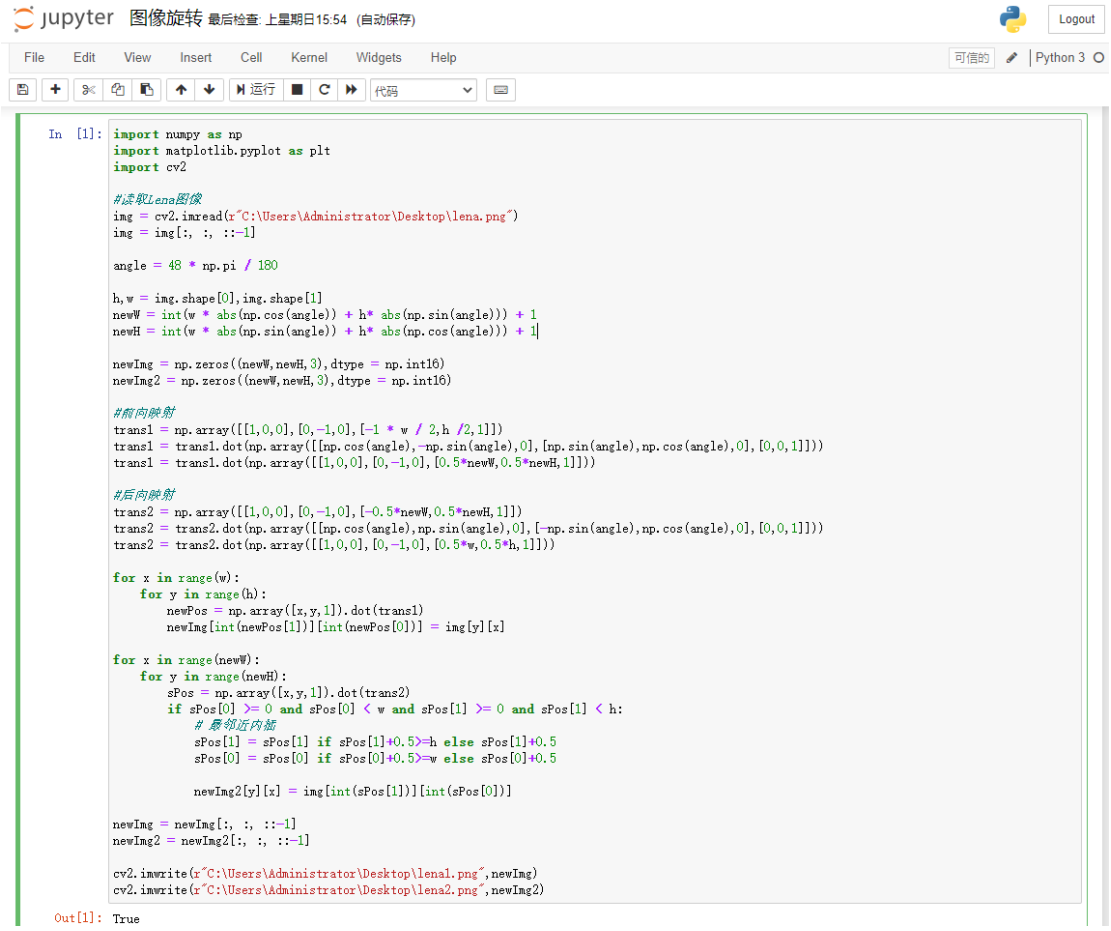
5. 实验效果

图 1

图 2

图 1 采用前向映射算法生产的旋转后的图像，明显可见图像中规律地存在部分像素的缺失。图 2 采用后向映射中的最邻近算法，则无像素缺失的情况存在。

6. 相关代码


jupyter 图像旋转 最后检查: 上星期日15:54 (自动保存)
Logout
File Edit View Insert Cell Kernel Widgets Help
可信的 Python 3
运行 代码
In [1]:
import numpy as np
import matplotlib.pyplot as plt
import cv2
#读取Lena图像
img = cv2.imread(r"C:\Users\Administrator\Desktop\lena.png")
img = img[:, :, ::-1]
angle = 48 * np.pi / 180
h,w = img.shape[0],img.shape[1]
newW = int(w * abs(np.cos(angle)) + h* abs(np.sin(angle))) + 1
newH = int(w * abs(np.sin(angle)) + h* abs(np.cos(angle))) + 1
newImg = np.zeros((newW,newH,3),dtype = np.int16)
newImg2 = np.zeros((newW,newH,3),dtype = np.int16)
#前向映射
trans1 = np.array([[1,0,0],[0,-1,0],[-1 * w / 2,h /2,1]])
trans1 = trans1.dot(np.array([[np.cos(angle),-np.sin(angle),0],[np.sin(angle),np.cos(angle),0],[0,0,1]]))
trans1 = trans1.dot(np.array([[1,0,0],[0,-1,0],[0.5*newW,0.5*newH,1]]))
#后向映射
trans2 = np.array([[1,0,0],[0,-1,0],[-0.5*newW,0.5*newH,1]])
trans2 = trans2.dot(np.array([[np.cos(angle),np.sin(angle),0],[-np.sin(angle),np.cos(angle),0],[0,0,1]]))
trans2 = trans2.dot(np.array([[1,0,0],[0,-1,0],[0.5*w,0.5*h,1]]))
for x in range(w):
for y in range(h):
newPos = np.array([x,y,1]).dot(trans1)
newImg[int(newPos[1])][int(newPos[0])] = img[y][x]
for x in range(newW):
for y in range(newH):
sPos = np.array([x,y,1]).dot(trans2)
if sPos[0] >= 0 and sPos[0] < w and sPos[1] >= 0 and sPos[1] < h:
# 最邻近内插
sPos[1] = sPos[1] if sPos[1]+0.5>=h else sPos[1]+0.5
sPos[0] = sPos[0] if sPos[0]+0.5>=w else sPos[0]+0.5
newImg2[y][x] = img[int(sPos[1])][int(sPos[0])]
newImg = newImg[:, :, ::-1]
newImg2 = newImg2[:, :, ::-1]
cv2.imwrite(r"C:\Users\Administrator\Desktop\lena1.png",newImg)
cv2.imwrite(r"C:\Users\Administrator\Desktop\lena2.png",newImg2)
Out[1]: True