TASK2 几何变换

一、几何空间变换基础

几何变换由两个基本操作组成：

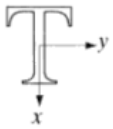
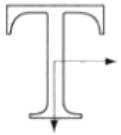
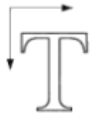
（1）坐标的空间变换；

（2）灰度内插，即对空间变换后的像素赋灰度值。

最常用的空间坐标变换之一是仿射变换，一般形式为：

$$[x \quad y \quad 1] = [v \quad w \quad 1]T = [v \quad w \quad 1]\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t32 & 1 \end{bmatrix}$$

(v,w)是原图像中的坐标，(x,y)是变换后图像中像素的坐标。根据下表选择对应的T矩阵即可得到对应的变换。

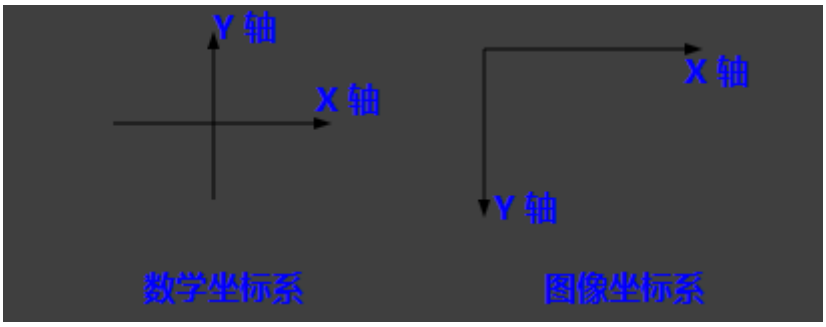
变换名称	仿射矩阵T	坐标公式	例子
恒等变换	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = w$	
尺度变换	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = c_x v$ $y = c_y w$	
旋转变换	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v\cos\theta - w\sin\theta$ $y = v\sin\theta + w\cos\theta$	
平移变换	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$	$x = v + t_x$ $y = w + t_y$	
（垂直）偏移变换	$\begin{bmatrix} 1 & 0 & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = vs_v + w$ $y = w$	
（水平）偏移变换	$\begin{bmatrix} 1 & s_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = s_h v + w$	

注意，这里的坐标系是笛卡尔坐标系，我们如果直接从图像坐标系变换，就需要先进行坐标系变换。

二、坐标系变换

变换中心，对于缩放、平移可以以图像坐标原点（图像左上角为原点）为中心变换，这不用坐标系变换，直接按照一般形式计算即可。而对于旋转和偏移，一般是以图像中心为原点，那么这就涉及坐标系转换了。

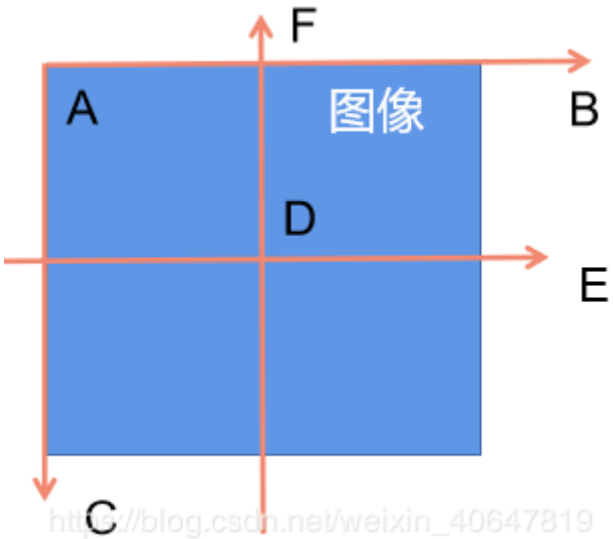
我们都知道，图像坐标的原点在图像左上角，水平向右为 X 轴，垂直向下为 Y 轴。数学课本中常见的坐标系是以图像中心为原点，水平向右为 X 轴，垂直向上为 Y 轴，称为笛卡尔坐标系。看下图:



因此，对于旋转和偏移，就需要3步（3次变换）：

- 将输入原图图像坐标转换为笛卡尔坐标系；
- 进行旋转计算。旋转矩阵前面已经给出了；
- 将旋转后的图像的笛卡尔坐标转回图像坐标。

图像坐标系与笛卡尔坐标系的转换关系，如图：



图像中的坐标系通常是AB和AC方向的,原点为A，而笛卡尔直角坐标系是DE和DF方向的，原点为D。 **令图像表示为M×N的矩阵，对于点A而言，两坐标系中的坐标分别是(0，0)和(-N/2,M/2)，则图像某像素点(x',y')转换为笛卡尔坐标（x,y）转换关系为，x为列，y为行：**

$$x = x' - N/2$$
$$y = -y' + M/2$$

逆变换为：

$$x' = x + N/2$$
$$y' = -y + M/2$$

根据前面说的3个步骤（3次变换），旋转(顺时针旋转)的变换形式就为，3次变换就有3个矩阵：

$$(x,y,1) = (x',y',1)T = (x',y',1)\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -0.5*N & 0.5*M & 1 \end{bmatrix}\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5*N & 0.5*M & 1 \end{bmatrix}$$

三、前向映射和反向映射

我们有两种基本方法来使用仿射变换的公式。

第一种方法称为**前向映射**，由扫描输入图像的像素，并在每个位置（v,w）用计算公式直接计算输出图像中相应像素的空间位置（x,y）组成。即根据原图用变换公式直接算出输出图像相应像素的空间位置。但是前向映射的问题是，可能输入图像的多个像素坐标映射到输出图像的同一位置，也可能输出图像的某些位置完全没有相应的输入图像像素与它匹配，也就是没有被映射到，造成有规律的空洞（黑色的蜂窝状）。

第二种方法是**反向映射**，扫描输出像素的位置，并在每一个位置（x,y）使用$(v,w)=T^{-1}(x,y)$计算输入图像中的相应位置，然后通过插值方法决定输出图像该位置的灰度值，一般采用双线性插值法。

关于向前和向后映射的具体步骤，详见此博客：https://blog.csdn.net/glorydream2015/article/details/44873703。

四、基于OpenCV的实现

基本的函数：

```
cv2.flip() # 图像翻转
cv2.warpAffine()  #图像仿射
cv2.getRotationMatrix2D()  #取得旋转角度的Matrix
cv2.GetAffineTransform(src, dst, mapMatrix) #取得图像仿射的matrix
cv2.getPerspectiveTransform(src, dst) #取得图像透视的 4 个点起止值
cv2.warpPerspective()  #图像透视
```

4.1 图像翻转

```
cv2.flip(src, flipCode[, dst]) → dst
```

- src: 原始图像矩阵；
- dst: 变换后的矩阵；
- flipCode: 翻转模式，有三种模式：
- 0 --- 垂直方向翻转； 1----- 水平方向翻转； -1：水平、垂直方向同时翻转

4.2 图像仿射

```
cv2.warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]]) → dst
```

- src – 输入图像；
- M – 变换矩阵；
- dsize – 输出图像的大小；
- flags – 插值方法的组合（int 类型！默认为线性插值）；
- borderMode – 边界像素模式（int 类型！）；
- borderValue – （重点！）边界填充值; 默认情况下，它为0。

4.3 图像旋转

```
cv2.getRotationMatrix2D(center, angle, scale)
```

- center：图片的旋转中心；
- angle：旋转角度；
- scale：缩放比例，如0.5表示缩小一半。

4.4 仿射变换

```
cv2.getAffineTransform(src, dst, mapMatrix) → None
```

- retval：返回值，仿射变换矩阵；
- src：原始图像，三个点；
- dst：目标图像，三个点。

4.5 图像透视

```
cv2.getPerspectiveTransform(src, dst)
```

- src：输入图像的四边形顶点坐标；
- dst：输出图像的相应的四边形顶点坐标。

```
cv2.warpPerspective(src, M, dsize[, dst[, flags[, borderMode[,
borderValue]]]]) → dst
```

- src – input image.
- dst – output image that has the size dsize and the same type as src .
- M – 3*3 transformation matrix.
- dsize – size of the output image.
- flags – combination of interpolation methods (INTER_LINEAR or INTER_NEAREST) and the optional flag
- WARP_INVERSE_MAP, that sets M as the inverse transformation (dst ---> src).
- borderMode – pixel extrapolation method (BORDER_CONSTANT or BORDER_REPLICATE).
- borderValue – value used in case of a constant border; by default, it equals 0.

对于视角变换，我们需要一个3x3变换矩阵。在变换前后直线还是直线。需要在原图上找到4个点，以及他们在输出图上对应的位置，这四个点中任意三个都不能共线，可以有函数cv2.getPerspectiveTransform()构建，然后这个矩阵传给函数cv2.warpPerspective()。

五、python的代码

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

def show(name, img):
    cv2.imshow(str(name), img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

img = cv2.imread('E:/PythonProgram/opencv_study/fig_transaction/yoona.jpg',
cv2.IMREAD_UNCHANGED)
rows, cols, channel = img.shape
# 1、平移操作
a = np.float32([[1, 0, 100], [0, 1, 50]])
dst = cv2.warpAffine(img, a, (cols, rows))
```

```
show('img', dst)
# 2、翻转操作
img_flip = cv2.flip(img, 0)
show('flip', img_flip)
# 3、旋转操作
rst = cv2.getRotationMatrix2D((cols/2, rows/2), 45, 0.6)
dst = cv2.warpAffine(img, rst, (cols, rows))
show('rotation', dst)
# 4、仿射变换  图像的旋转加上拉升就是图像仿射变换
pts1 = np.float32([[50,50], [200,50], [50,200]])
pts2 = np.float32([[10,100], [200,50], [100,250]])
M = cv2.getAffineTransform(pts1, pts2)
dst = cv2.warpAffine(img, M, (cols,rows))
plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()  # 直接使用plt输出的话会颜色差别很大，因为opencv的接口使用BGR，而matplotlib.pyplot 则是RGB模式
"""
代码cv2读入的是BGR模式，在opencv里面存储的是BGR，所以img用opencv输出就是正常颜色；
而matplotlib.pyplot是RGB模式，当用cv读入，直接用matplotlib.pyplot输出，颜色就变了，所以需要调整颜色的顺序，就变成了img2；
"""
# 下面的代码解决了这个问题
b, g, r = cv2.split(img)
img2 = cv2.merge([r, g, b])
pts1 = np.float32([[50,50], [200,50], [50,200]])
pts2 = np.float32([[10,100], [200,50], [100,250]])
M = cv2.getAffineTransform(pts1, pts2)
dst = cv2.warpAffine(img2, M, (cols,rows))
plt.subplot(121),plt.imshow(img2),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
# 5、透视变换
pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])
M=cv2.getPerspectiveTransform(pts1,pts2)
dst=cv2.warpPerspective(img2,M,(300,300))
plt.subplot(121),plt.imshow(img2),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```