[Python图像处理] 三十六.OpenCV图像几何变换万字详解（平移缩放旋转、镜像仿射透视）

原创 Eastmount 2021-02-01 22:08:06 4455 收藏 38 编辑 版权

分类专栏： Python图像处理及图像识别 文章标签： Python图像处理 数字图像处理 OpenCV 几何变换 图像校正

该系列文章是讲解Python OpenCV图像处理知识，前期主要讲解图像入门、OpenCV基础用法，中期讲解图像处理的各种算法，包括图像锐化算子、图像增强技术、图像分割等，后期结合深度学习研究图像识别、图像分类应用。希望文章对您有所帮助，如果有不足之处，还请海涵~

前面一篇文章介绍了OpenCV图像处理入门知识、算数逻辑运算与图像融合。这篇文章将介绍图像几何变换，包括：图像平移变换、图像缩放变换、图像旋转变换、图像镜像变换、图像仿射变换和图像透视变换，万字长文整理，希望对您有所帮助。 同时，该部分知识均为作者查阅资料撰写总结，并且开设成了收费专栏，为小宝赚点奶粉钱，感谢您的抬爱。当然如果您是在读学生或经济拮据，可以私聊我给你每篇文章开白名单，或者转发原文给你，更希望您能进步，一起加油喔~

- https://github.com/eastmountyxz/ImageProcessing-Python

文章目录

前文参考：

- [Python图像处理] 一.图像处理基础知识及OpenCV入门函数
- [Python图像处理] 二.OpenCV+Numpy库读取与修改像素
- [Python图像处理] 三.获取图像属性、兴趣ROI区域及通道处理
- [Python图像处理] 四.图像平滑之均值滤波、方框滤波、高斯滤波及中值滤波
- [Python图像处理] 五.图像融合、加法运算及图像类型转换
- [Python图像处理] 六.图像缩放、图像旋转、图像翻转与图像平移
- [Python图像处理] 七.图像阈值化处理及算法对比
- [Python图像处理] 八.图像腐蚀与图像膨胀
- [Python图像处理] 九.形态学之图像开运算、闭运算、梯度运算
- [Python图像处理] 十.形态学之图像顶帽运算和黑帽运算
- [Python图像处理] 十一.灰度直方图概念及OpenCV绘制直方图

- [Python图像处理] 十二.图像几何变换之图像仿射变换、图像透视变换和图像校正
- [Python图像处理] 十三.基于灰度三维图的图像顶帽运算和黑帽运算
- [Python图像处理] 十四.基于OpenCV和像素处理的图像灰度化处理
- [Python图像处理] 十五.图像的灰度线性变换
- [Python图像处理] 十六.图像的灰度非线性变换之对数变换、伽马变换
- [Python图像处理] 十七.图像锐化与边缘检测之Roberts算子、Prewitt算子、Sobel算子和Laplacian算子
- [Python图像处理] 十八.图像锐化与边缘检测之Scharr算子、Canny算子和LOG算子
- [Python图像处理] 十九.图像分割之基于K-Means聚类的区域分割
- [Python图像处理] 二十.图像量化处理和采样处理及局部马赛克特效
- [Python图像处理] 二十一.图像金字塔之图像向下取样和向上取样
- [Python图像处理] 二十二.Python图像傅里叶变换原理及实现
- [Python图像处理] 二十三.傅里叶变换之高通滤波和低通滤波
- [Python图像处理] 二十四.图像特效处理之毛玻璃、浮雕和油漆特效
- [Python图像处理] 二十五.图像特效处理之素描、怀旧、光照、流年以及滤镜特效
- [Python图像处理] 二十六.图像分类原理及基于KNN、朴素贝叶斯算法的图像分类案例
- [Python图像处理] 二十七.OpenGL入门及绘制基本图形（一）
- [Python图像处理] 二十八.OpenCV快速实现人脸检测及视频中的人脸
- [Python图像处理] 二十九.MoviePy视频编辑库实现抖音短视频剪切合并操作
- [Python图像处理] 三十.图像量化及采样处理万字详细总结（推荐）
- [Python图像处理] 三十一.图像点运算处理两万字详细总结（灰度化处理、阈值化处理）
- [Python图像处理] 三十二.傅里叶变换（图像去噪）与霍夫变换（特征识别）万字详细总结
- [Python图像处理] 三十三.图像各种特效处理及原理万字详解（毛玻璃、浮雕、素描、怀旧、流年、滤镜等）
- [Python图像处理] 三十四.数字图像处理基础与几何图形绘制万字详解（推荐）
- [Python图像处理] 三十五.OpenCV图像处理入门、算数逻辑运算与图像融合（推荐）
- [Python图像处理] 三十六.OpenCV图像几何变换万字详解（平移缩放旋转、镜像仿射透视）

图像几何变换又称为图像空间变换，是各种图像处理算法的基础。它是在不改变图像内容的情况下，对图像像素进行空间几何变换的处理方式。它将一幅图像中的坐标位置映射到另一幅图像中的新坐标位置，其实质是改变像素的空间位置，估算新空间位置上的像素值。

一.图像几何变换概述

图像几何变换不改变图像的像素值，只是在图像平面上进行像素的重新安排。适当的几何变换可以最大程度地消除由于成像角度、透视关系乃至镜头自身原因所造成的几何失真所产生的负面影响。几何变换常常作为图像处理应用的预处理步骤，是图像归一化的核心工作之一。

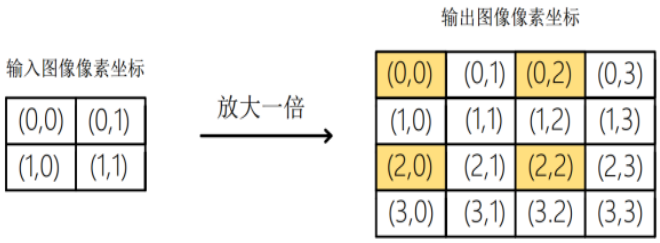
一个几何变换需要两部分运算：首先是空间变换所需的运算，如平移、缩放、旋转和正平行投影等， 需要用它来表示输出图像与输入图像之间的像素映射关系；此外，还需要使用灰度插值算法， 因为按照这种变换关系进行计算， 输出图像的像素可能被映射到输入图像的非整数坐标上。

图像的几何变换主要包括：

- **图形平移**
- **图像缩放**
- **图像旋转**

- **图像镜像**
- **图像仿射**
- **图像透视**
- …

图像变换是建立在矩阵运算基础上的，通过矩阵运算可以很快的找到对应关系。图像几何变换在变换过程中会建立一种原图像像素与变换后图像像素之间的映射关系，通过这种关系，能够从一方的像素计算出另一方的像素的坐标位置。通常将图像坐标映射到输出的过程称作向前映射，反之，将输出图像映射到输入的过程称作向后映射。向后映射在实践中使用较多，原因是能够避免使用向前映射中出现映射不完全和映射重叠的问题。图1展示了图像放大的示例，右边图中只有(0,0)、(0,2)、(2,0)、(2,2)四个坐标根据映射关系在原图像中找到了相对应的像素，其余的12个坐标没有有效值。

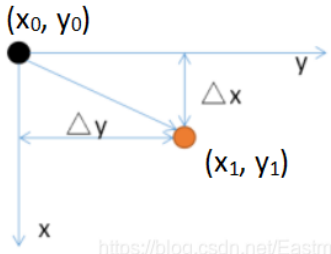


对于数字图像而言，像素的坐标是离散型非负整数，但是在进行变换的过程中有可能产生浮点坐标值。这在图像处理中是一个无效的坐标。为了解决这个问题需要用到插值算法。常见算法有最近邻插值法、双线性插值法、双立方插值法等。

- **最近邻插值**
  浮点坐标的像素值等于距离该店最近的输入图像的像素值。
- **双线性插值**
  主要思想是计算出浮点坐标像素近似值。一个浮点坐标必定会被四个整数坐标所包围，将这个四个整数坐标的像素值按照一定的比例混合就可以求出浮点坐标的像素值。
- **双立方插值**
  双立方插值是一种更加复杂的插值方式，它能创造出比双线性插值更平滑的图像边缘。在图像处理中，双立方插值计算涉及到周围16个像素点，插值后的坐标点是原图中邻近16个像素点的权重卷积之和。

二.图像平移变换

图像平移是将图像中的所有像素点按照给定的平移量进行水平或垂直方向上的移动。假设原始像素的位置坐标为（x0，y0），经过平移量（△x，△y）后，坐标变为（x1, y1），如图2所示。



用数学式子表示为公式（1）。

$$x_1 = x_0 + \Delta x$$
$$y_1 = y_0 + \Delta y$$

用矩阵表示如公式（2）所示，式子中，矩阵称为平移变换矩阵或因子，△x和△y称为平移量。

$$[x_1 \quad y_1 \quad 1] = [x_0 \quad y_0 \quad 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{bmatrix}$$

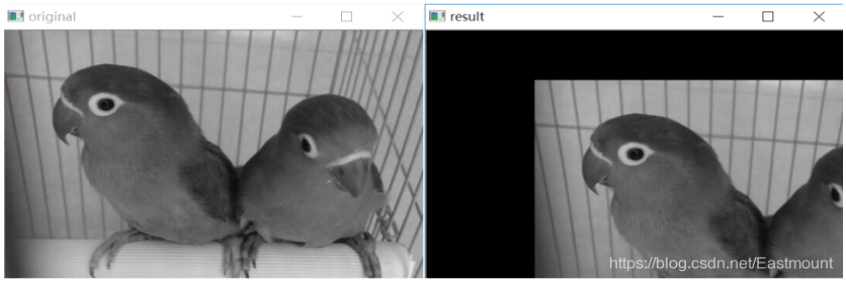
图像平移首先定义平移矩阵M，再调用warpAffine()函数实现平移，核心函数如下：

- **M = np.float32([[1, 0, x], [0, 1, y]])**
  M表示平移矩阵，其中x表示水平平移量，y表示垂直平移量
- **shifted = cv2.warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])**
  – src表示原始图像
  – M表示平移矩阵
  – dsize表示变换后的输出图像的尺寸大小
  – dst为输出图像，其大小为dsize，类型与src相同
  – flag表示插值方法的组合和可选值
  – borderValue表示像素外推法，当borderMode = BORDER_TRANSPARENT时，表示目标图像中的像素不会修改源图像中的“异常值”。
  – borderValue用于边界不变的情况，默认情况下为0

下面代码是图像平移的一个简单案例，它定义了图像平移矩阵M，然后调用warpAffine()函数将原始图像垂直向下平移了50个像素，水平向右平移了100个像素。

```
#encoding:utf-8
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np

#读取图片
src = cv2.imread('test.bmp')

#图像平移矩阵
M = np.float32([[1, 0, 100], [0, 1, 50]])

#获取原始图像列数和行数
rows, cols = src.shape[:2]

#图像平移
result = cv2.warpAffine(src, M, (cols, rows))

#显示图像
cv2.imshow("original", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

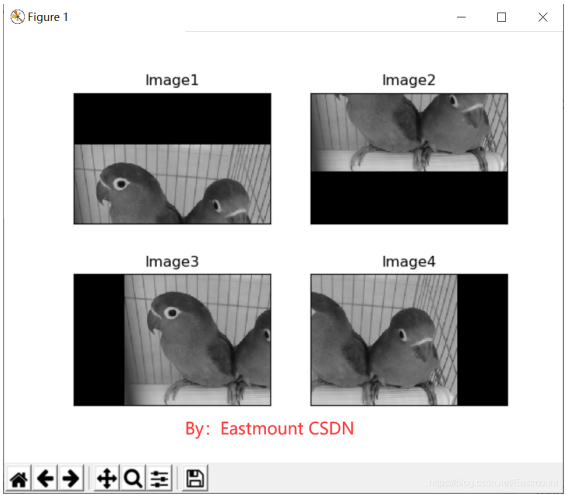
输出结果如图3所示：



下面一个案例是将图像分别向下、向上、向右、向左平移，再调用matplotlib绘图库依次绘制的过程。

```
#encoding:utf-8
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
img = cv2.imread('test.bmp')
image = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#图像平移
#垂直方向 向下平移100
M = np.float32([[1, 0, 0], [0, 1, 100]])
img1 = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

#垂直方向 向上平移100
M = np.float32([[1, 0, 0], [0, 1, -100]])
img2 = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

#水平方向 向右平移100
M = np.float32([[1, 0, 100], [0, 1, 0]])
img3 = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

#水平方向 向左平移100
M = np.float32([[1, 0, -100], [0, 1, 0]])
img4 = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

#循环显示图形
titles = [ 'Image1', 'Image2', 'Image3', 'Image4']
images = [img1, img2, img3, img4]
for i in range(4):
   plt.subplot(2,2,i+1), plt.imshow(images[i], 'gray')
   plt.title(titles[i])
   plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图4所示，它从四个方向都进行了平移，并且调用subplot()函数将四个子图绘制在一起。



三.图像缩放变换

图像缩放（image scaling）是指对数字图像的大小进行调整的过程。在Python中，图像缩放主要调用resize()函数实现，函数原型如下：

- **result = cv2.resize(src, dsize[, result[. fx[, fy[, interpolation]]]])**
  – src表示原始图像
  – dsize表示图像缩放的大小
  – result表示图像结果
  – fx表示图像x轴方向缩放大小的倍数
  – fy表示图像y轴方向缩放大小的倍数
  – interpolation表示变换方法。CV_INTER_NN表示最近邻插值；CV_INTER_LINEAR表示双线性插值（缺省使用）；CV_INTER_AREA表示使用像素关系重采样，当图像缩小时，该方法可以避免波纹出现，当图像放大时，类似于CV_INTER_NN；CV_INTER_CUBIC表示立方插值

常见的图像缩放两种方式如下所示，第一种方式是将原图像设置为(160, 160)像素大小，第二种方式是将原始图像缩小为0.5倍。

- result = cv2.resize(src, (160,160))
- result = cv2.resize(src, None, fx=0.5, fy=0.5)

设(x1, y1)是缩放后的坐标，(x0, y0)是缩放前的坐标，sx、sy为缩放因子，则图像缩放的计算公式（3）所示：

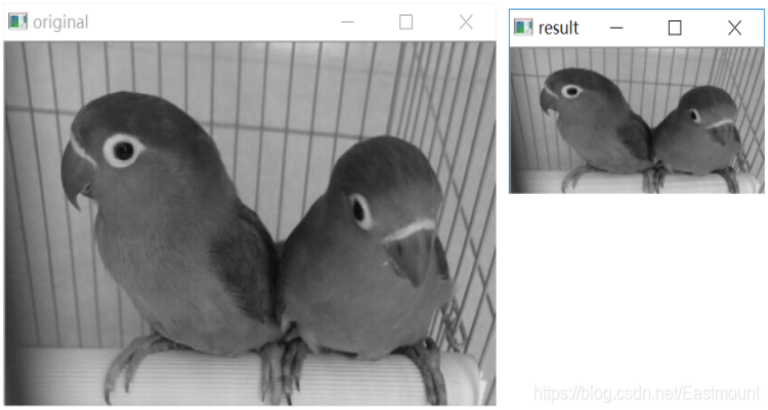
$$[x_1 \quad y_1 \quad 1] = [x_0 \quad y_0 \quad 1] \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

下面是Python实现图像缩放的代码，它将所读取的test.bmp图像进行缩小。

```
#encoding:utf-8
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np

#读取图片
src = cv2.imread('test.bmp')

#图像缩放
result = cv2.resize(src, (200,100))
```

```
print(result.shape)

#显示图像
cv2.imshow("original", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图5所示，图像缩小为（200，100）像素。注意，代码中调用函数 cv2.resize(src, (200,100)) 设置新图像大小dsize的列数为200，行数为100。



下面讲解另一种图像缩放变换的方法，通过原始图像像素乘以缩放系数进行图像变换，代码如下：

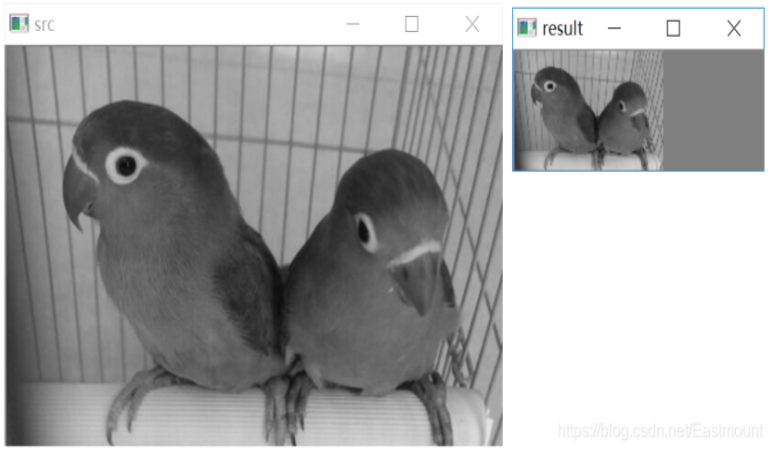
```
#encoding:utf-8
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np

#读取图片
src = cv2.imread('test.bmp')
rows, cols = src.shape[:2]
print(rows, cols)

#图像缩放 dsize(列,行)
result = cv2.resize(src, (int(cols*0.6), int(rows*1.2)))

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

获取图片“test.bmp”的元素像素值，其rows值为250，cols值为387，接着进行宽度缩小0.6倍、高度放大1.2倍的处理，运行前后对比效果如图6所示。

最后讲解调用(fx,fy)参数设置缩放倍数的方法，对原始图像进行放大或缩小操作。下面代码是fx和fy方向缩小至原始图像0.3倍的操作。

```
#encoding:utf-8
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np

#读取图片
src = cv2.imread('test.bmp')
rows, cols = src.shape[:2]
print(rows, cols)

#图像缩放
result = cv2.resize(src, None, fx=0.3, fy=0.3)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

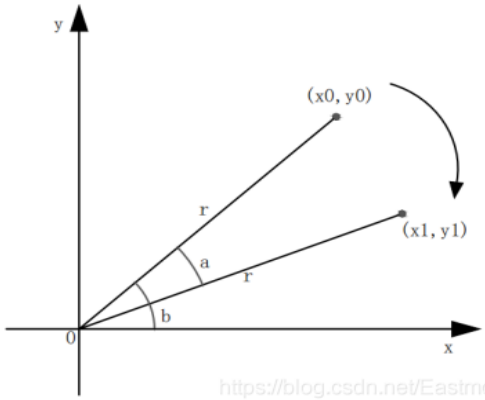
输出的结果如图7所示，这是按比例0.3×0.3缩小的。



四.图像旋转变换

图像旋转是指图像以某一点为中心旋转一定的角度，形成一幅新的图像的过程。图像旋转变换会有一个旋转中心，这个旋转中心一般为图

像的中心，旋转之后图像的大小一般会发生改变。图8表示原始图像的坐标(x0, y0)旋转至(x1, y1)的过程。



旋转公式如（4）所示，其中(m,n)是旋转中心，a是旋转的角度，(left,top)是旋转后图像的左上角坐标。

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -m & n & 1 \end{bmatrix} \begin{bmatrix} \cos a & -\sin a & 0 \\ \sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ left & top & 1 \end{bmatrix}$$

图像旋转变换主要调用getRotationMatrix2D()函数和warpAffine()函数实现，绕图像的中心旋转，函数原型如下：

- **M = cv2.getRotationMatrix2D(center, angle, scale)**
  – center表示旋转中心点，通常设置为(cols/2, rows/2)
  – angle表示旋转角度，正值表示逆时针旋转，坐标原点被定为左上角
  – scale表示比例因子

- **rotated = cv2.warpAffine(src, M, (cols, rows))**
  – src表示原始图像
  – M表示旋转参数，即getRotationMatrix2D()函数定义的结果
  – (cols, rows)表示原始图像的宽度和高度

实现代码如下所示：

```
#encoding:utf-8
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np

#读取图片
src = cv2.imread('test.bmp')

#源图像的高、宽 以及通道数
rows, cols, channel = src.shape

#绕图像的中心旋转
M = cv2.getRotationMatrix2D((cols/2, rows/2), 30, 1) #旋转中心 旋转度数 scale
rotated = cv2.warpAffine(src, M, (cols, rows))  #原始图像 旋转参数 元素图像宽高

#显示图像
cv2.imshow("src", src)
cv2.imshow("rotated", rotated)

#等待显示
```

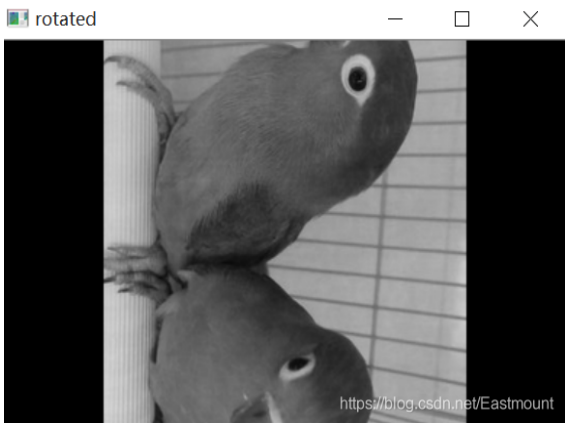
```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

显示效果如图9所示，绕图像中心点逆时针旋转30度。

如果设置的旋转度数为负数，则表示顺时针旋转，下述代码表示将图像顺时针旋转90度。

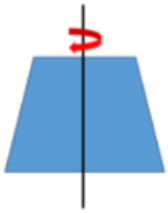
```
#encoding:utf-8
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np

#读取图片
src = cv2.imread('test.bmp')

#源图像的高、宽 以及通道数
rows, cols, channel = src.shape

#绕图像的中心旋转
#函数参数：旋转中心 旋转度数 scale
M = cv2.getRotationMatrix2D((cols/2, rows/2), -90, 1)

#函数参数：原始图像 旋转参数 元素图像宽高
rotated = cv2.warpAffine(src, M, (cols, rows))

#显示图像
cv2.imshow("src", src)
cv2.imshow("rotated", rotated)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

旋转之后的图像如图10所示。



五.图像镜像变换

图像镜像变换是图像旋转变换的一种特殊情况，通常包括垂直方向和水平方向的镜像。水平镜像通常是以原图像的垂直中轴为中心，将图像分为左右两部分进行堆成变换。如图11所示：



垂直镜像通常是以原图像的水平中轴线为中心，将图像划分为上下两部分进行堆成变换的过程，示意图如图12所示。

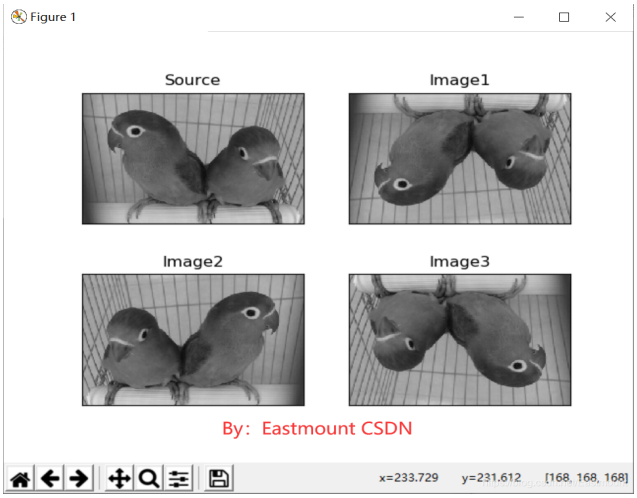
在Python中主要调用OpenCV的flip()函数实现图像镜像变换，函数原型如下：

dst = cv2.flip(src, flipCode)
– src表示原始图像
– flipCode表示翻转方向，如果flipCode为0，则以X轴为对称轴翻转，如果fliipCode>0则以Y轴为对称轴翻转，如果flipCode<0则在X轴、Y轴方向同时翻转。

下面的代码是实现三个方向的翻转。

```
#encoding:utf-8
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
img = cv2.imread('test.bmp')
src = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#图像翻转
img1 = cv2.flip(src, 0)   #参数=0以X轴为对称轴翻转
img2 = cv2.flip(src, 1)   #参数>0以Y轴为对称轴翻转
img3 = cv2.flip(src, -1)  #参数<0X轴和Y轴翻转

#显示图形
titles = ['Source', 'Image1', 'Image2', 'Image3']
images = [src, img1, img2, img3]
for i in range(4):
   plt.subplot(2,2,i+1), plt.imshow(images[i], 'gray')
   plt.title(titles[i])
   plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如图13所示，图中“Source”为原始图像，“Image1”为以X轴为对称轴翻转或垂直镜像，“Image2”为以Y轴为对称轴翻转或水平镜像，“Images3”为以X轴和Y轴翻转。

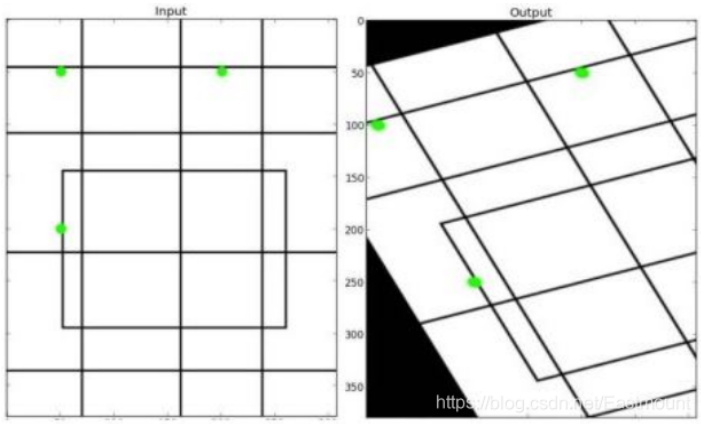


六.图像仿射变换

图像仿射变换又称为图像仿射映射，是指在几何中，一个向量空间进行一次线性变换并接上一个平移，变换为另一个向量空间。通常图像的旋转加上拉升就是图像仿射变换，仿射变换需要一个M矩阵实现，但是由于仿射变换比较复杂，很难找到这个M矩阵，OpenCV提供了根据变换前后三个点的对应关系来自动求解M的函数：

- cv2.getAffineTransform(pos1,pos2)

其中pos1和pos2表示变换前后的对应位置关系，输出的结果为仿射矩阵M，接着使用函数cv2.warpAffine()实现图像仿射变换。图14是仿射变换的前后效果图。



图像仿射变换的函数原型如下：

- **M = cv2.getAffineTransform(pos1,pos2)**
  – pos1表示变换前的位置
  – pos2表示变换后的位置

- **cv2.warpAffine(src, M, (cols, rows))**
  – src表示原始图像
  – M表示仿射变换矩阵
  – (rows,cols)表示变换后的图像大小，rows表示行数，cols表示列数

实现代码如下所示：

```
#encoding:utf-8
```

```python
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
src = cv2.imread('test.bmp')

#获取图像大小
rows, cols = src.shape[:2]

#设置图像仿射变换矩阵
pos1 = np.float32([[50,50], [200,50], [50,200]])
pos2 = np.float32([[10,100], [200,50], [100,250]])
M = cv2.getAffineTransform(pos1, pos2)

#图像仿射变换
result = cv2.warpAffine(src, M, (cols, rows))

#显示图像
cv2.imshow("original", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

输出结果如图15所示:

七.图像透视变换

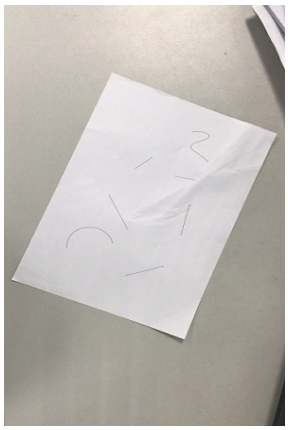
图像透视变换（Perspective Transformation）的本质是将图像投影到一个新的视平面，同理OpenCV通过函数cv2.getPerspectiveTransform(pos1,pos2)构造矩阵M，其中pos1和pos2分别表示变换前后的4个点对应位置。得到M后在通过函数cv2.warpPerspective(src,M,(cols,rows))进行透视变换。

图像透视变换的函数原型如下:

- **M = cv2.getPerspectiveTransform(pos1, pos2)**
  – pos1表示透视变换前的4个点对应位置
  – pos2表示透视变换后的4个点对应位置

- **cv2.warpPerspective(src,M,(cols,rows))**
  – src表示原始图像
  – M表示透视变换矩阵
  – (rows,cols)表示变换后的图像大小，rows表示行数，cols表示列数

假设现在存在一张A4纸图像，现在需要通过调用图像透视变换校正图像。

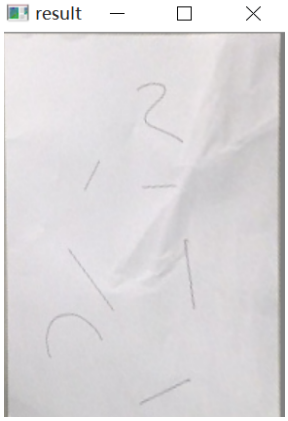


图像透视变换的校正代码如下所示，代码中pos1表示透视变换前A4纸的四个顶点，pos2表示透视变换后A4纸的四个顶点。

```
# -*- coding: utf-8 -*-
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
src = cv2.imread('test01.jpg')

#获取图像大小
rows, cols = src.shape[:2]

#设置图像透视变换矩阵
pos1 = np.float32([[114, 82], [287, 156], [8, 322], [216, 333]])
pos2 = np.float32([[0, 0], [188, 0], [0, 262], [188, 262]])
M = cv2.getPerspectiveTransform(pos1, pos2)

#图像透视变换
result = cv2.warpPerspective(src, M, (190, 272))

#显示图像
cv2.imshow("original", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

最终输出结果如图17所示，它将图形校正显示。

- (8, 332) (114, 82)
- (207, 357) (287, 157)



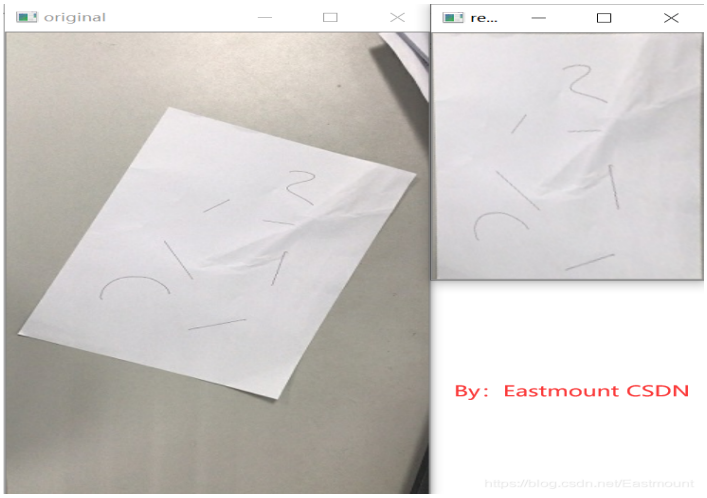
注意，上面的顶点是自定义，接下来的代码实现自动校正。步骤如下：

- 获取图像大小
- 源图像高斯模糊
- 进行灰度化处理
- 边缘检测（检测出图像的边缘信息）
- 通过霍夫变换得到A4纸边缘
- 输出的四个点分别为四个顶点
- 绘制边缘
- 根据四个顶点设置图像透视变换矩阵
- 图像透视变换
- 显示图像

```
#encoding:utf-8
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
src = cv2.imread('test01.jpg')

#获取图像大小
rows, cols = src.shape[:2]

#将源图像高斯模糊
img = cv2.GaussianBlur(src, (3,3), 0)
#进行灰度化处理
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

#边缘检测（检测出图像的边缘信息）
edges = cv2.Canny(gray,50,250,apertureSize = 3)
cv2.imwrite("canny.jpg", edges)

#通过霍夫变换得到A4纸边缘
lines = cv2.HoughLinesP(edges,1,np.pi/180,50,minLineLength=90,maxLineGap=10)

#下面输出的四个点分别为四个顶点
for x1,y1,x2,y2 in lines[0]:
    print((x1,y1),(x2,y2))
for x1,y1,x2,y2 in lines[1]:
```

```
    print((x1,y1),(x2,y2))

#绘制边缘
for x1,y1,x2,y2 in lines[0]:
    cv2.line(gray, (x1,y1), (x2,y2), (0,0,255), 1)

#根据四个顶点设置图像透视变换矩阵
pos1 = np.float32([[114, 82], [287, 156], [8, 322], [216, 333]])
pos2 = np.float32([[0, 0], [188, 0], [0, 262], [188, 262]])
M = cv2.getPerspectiveTransform(pos1, pos2)

#图像透视变换
result = cv2.warpPerspective(src, M, (190, 272))

#显示图像
cv2.imshow("original", src)
cv2.imshow("result", result)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

图像透视变换校正结果如下图所示：



八.总结

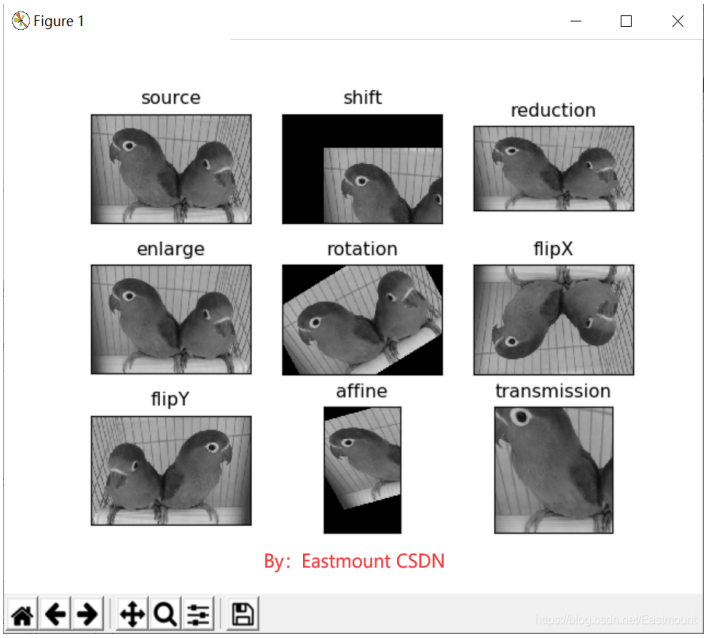
本文主要讲解了图像几何变换，通过原理和代码分别介绍了常见的六种几何变换，包括图像平移变换、图像缩放变换、图像旋转变换、图像镜像变换、图像仿射变换和图像透视变换。

各个算法对比完整代码如下：

```
#encoding:utf-8
#By:Eastmount CSDN 2021-02-01
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图片
img = cv2.imread('test.bmp')
image = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#图像平移矩阵
```

```
M = np.float32([[1, 0, 100], [0, 1, 80]])
rows, cols = image.shape[:2]
img1 = cv2.warpAffine(image, M, (cols, rows))

#图像缩小
img2 = cv2.resize(image, (200,100))

#图像放大
img3 = cv2.resize(image, None, fx=1.1, fy=1.1)

#绕图像的中心旋转
#源图像的高、宽 以及通道数
rows, cols, channel = image.shape
#函数参数：旋转中心 旋转度数 scale
M = cv2.getRotationMatrix2D((cols/2, rows/2), 30, 1)
#函数参数：原始图像 旋转参数 元素图像宽高
img4 = cv2.warpAffine(image, M, (cols, rows))

#图像翻转
img5 = cv2.flip(image, 0)   #参数=0以X轴为对称轴翻转
img6 = cv2.flip(image, 1)   #参数>0以Y轴为对称轴翻转

#图像的仿射
pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])
M = cv2.getAffineTransform(pts1,pts2)
img7 = cv2.warpAffine(image, M, (rows,cols))

#图像的透射
pts1 = np.float32([[56,65],[238,52],[28,237],[239,240]])
pts2 = np.float32([[0,0],[200,0],[0,200],[200,200]])
M = cv2.getPerspectiveTransform(pts1,pts2)
img8 = cv2.warpPerspective(image,M,(200,200))


#循环显示图形
titles = [ 'source', 'shift', 'reduction', 'enlarge', 'rotation', 'flipX', 'flipY', 'affine', 'transmissi
on']
images = [image, img1, img2, img3, img4, img5, img6, img7, img8]
for i in range(9):
   plt.subplot(3, 3, i+1), plt.imshow(images[i], 'gray')
   plt.title(titles[i])
   plt.xticks([]),plt.yticks([])
plt.show()
```

输出结果如下图所示：



- 源代码下载地址，记得帮忙点star和关注喔。
  https://github.com/eastmountyxz/ImageProcessing-Python

时光嘀嗒嘀嗒的流失，这是我在CSDN写下的第八篇年终总结，比以往时候来的更早一些。《敏而多思，宁静致远》，仅以此篇纪念这风雨兼程的一年，这感恩的一年。列车上只写了一半，这两天完成，思远，思君O(∩_∩)O

- 2020年总结：敏而多思，宁静致远——纪念这风雨兼程的一年

2020年8月18新开的“娜璋AI安全之家”，主要围绕Python大数据分析、网络空间安全、人工智能、Web渗透及攻防技术进行讲解，同时分享CCF、SCI、南核北核论文的算法实现。娜璋之家会更加系统，并重构作者的所有文章，从零讲解Python和安全，写了近十年文章，真心想把自己所学所感所做分享出来，还请各位多多指教，真诚邀请您的关注！谢谢。

(By:Eastmount 2021-02-01 晚上11点 http://blog.csdn.net/eastmount/)

参考文献，在此感谢这些大佬，共勉！

- [1] 罗子江, 杨秀璋. Python中的图像处理[M]. 2020.
- [2] 冈萨雷斯. 数字图像处理（第3版）[M]. 北京：电子工业出版社, 2013.
- [3] 阮秋琦. 数字图像处理学（第3版）[M]. 北京：电子工业出版社，2008.
- [4] 毛星云, 冷雪飞. OpenCV3编程入门[M]. 北京：电子工业出版社, 2015.
- [5] Eastmount. [Python图像处理] 六.图像缩放、图像旋转、图像翻转与图像平移[EB/OL]. (2018-09-06). https://blog.csdn.net/Eastmount/article/details/82454335.
- [6] Eastmount. [数字图像处理] 六.MFC空间几何变换之图像平移、镜像、旋转、缩放详解[EB/OL]. (2015-06-04).

https://blog.csdn.net/Eastmount/article/details/46345299.

- [7] on2way. Python下opencv使用笔记（三）（图像的几何变换）[EB/OL]. (2015-07-08). https://blog.csdn.net/on2way/article/details/46801063.
- [8] ywxk1314. 数字图像处理——图像的几何变换[EB/OL]. (2018-03-24). https://blog.csdn.net/ywxk1314/article/details/79681754.
- [9] 网易云课堂_高登教育. Python+OpenCV图像处理[EB/OL]. (2019-01-15). https://study.163.com/course/courseLearn.htm?courseId=1005317018.
- [10] t6_17. 图像校正-透视变换[EB/OL]. (2017-12-06). https://blog.csdn.net/t6_17/article/details/78729097.