第 5 章 几何变换

图像在获取过程中受到多种因素的影响，由于实际获取的图像和预期图像存在差异，因此需要对图像进行调整。例如获取的目标存在偏斜，通过对图像进行透视变换可以矫正图像。当用图像训练模型时，通过对图像进行变换可以获取更多图像特征、样本量。图像的几何变换就是指在不改变图像原有内容的基础上，对图像的像素空间位置进行改变，以达到变换图像中像素位置的目的。图像的几何变换包括图像缩放、翻转、平移、错切、旋转、仿射变换、透视变换。

5.1 图像缩放

图像缩放(Image Scaling)是指调整数字图像的尺寸。图像缩放既会改变图像的大小，也会影响图像的清晰度和平滑度。

5.1.1 基本原理

图像缩放是对图像像素坐标进行映射，已知 s_x、s_y 为缩放系数，原图像素坐标(x_0,y_0)与缩放后像素坐标(x,y)的关系如下：

$$\begin{cases} x = s_x \cdot x_0 \\ y = s_y \cdot y_0 \end{cases} \tag{5-1}$$

用矩阵表示：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{5-2}$$

图像放缩有多种方法，常用的有最近邻法、双线性插值法。

1. 最近邻

最近邻缩放的原理是根据缩放系数确定新图像每个位置上的像素。已知原图的尺寸为(h,w)，缩放后图像的尺寸为(nh,nw)，缩放系数为$\left(\frac{\mathrm{nh}}{h},\frac{\mathrm{nw}}{w}\right)$，用新图像坐标除以缩放系数

便可得到原图对应的坐标，如图 5-1 所示，原图的尺寸为(6,6)，把原图宽和高缩小为原来的二分之一，缩放后图像比例为(3,3)，缩放系数为$\left(\frac{3}{6},\frac{3}{6}\right)$，新图像坐标(0,0)对应的原图坐标为$\left(\frac{0}{3/6},\frac{0}{3/6}\right)$。新图像坐标(3,3)对应原图$\left(\frac{3}{3/6},\frac{3}{3/6}\right)$坐标的像素。以此类推，得到新图像所有元素的像素，用最近邻放大图像也是同样的原理。

	0	1	2	3	4	5	6
0	100	89	23	45	50	46	20
1	70	156	67	203	87	82	43
2	30	145	32	126	86	62	67
3	60	142	24	56	72	120	78
4	89	89	56	67	75	78	74
5	110	98	36	32	66	38	98
6	120	72	87	87	65	42	10

(a) 原图

	0	1	2	3
0	100	23	50	20
1	30	32	86	67
2	89	56	75	74
3	120	87	65	10

(b) 缩小

图 5-1　最近邻法缩放示例

【例 5-1】 最近邻法缩放图像。

解：

(1) 读取彩色图像。

(2) 最近邻缩放。

(3) 显示图像，代码如下：

```
#chapter5_1.py
import cv2
import numpy as np
import matplotlib.pyplot as plt

def neighbor_size(img,nh,nw):
    '''
    最近邻缩放
    :param img: 读取彩色图像
    :param nh: 新设置的图像高度,new_height
    :param nw: 新设置的图像宽度,new_width
    :return: 新图像
    '''
    h,w,c = img.shape
    #新图像的尺寸
    img_new = np.zeros([nh, nw, c], np.uint8)
    #缩放系数
    h_scale = nh *1.0 / h
    w_scale = nw *1.0 / w
    #计算新图像上每个位置对应的像素
```

```
        for i in range(nh):
            for j in range(nw):
                img_new[i, j] = img[int(i / h_scale), int(j / w_scale)]
        return img_new

    if __name__ == '__main__':
        #1. 读取彩色图像
        img = cv2.imread('pictures/L1.png', 1)
        #2. 最近邻缩放
        #缩小
        img1 = neighbor_size(img,nh=40,nw=80)
        #放大
        img2 = neighbor_size(img,nh=1600,nw=800)
        #3. 显示图像
        plt.subplot(131)
        plt.axis('off')
        plt.imshow(img2[..., ::-1])
        plt.subplot(132)
        plt.axis('off')
        plt.imshow(img[..., ::-1])
        plt.subplot(133)
        plt.axis('off')
        plt.imshow(img1[..., ::-1])
```

运行结果如图 5-2 所示。

(a) 放大　　(b) 原图　　(c) 缩小

图 5-2　最近邻法缩放图像

2. 双线性插值法

在最近邻法中，根据缩放系数计算新图坐标在原图上的位置可能不是整数。例如缩放系数为 2，新图坐标(7,7)对应原图(3.5,3.5)的位置，最近邻法采取取整方法得到(3,3)，这种方法简单，但是会损失一部分图像信息，而插值法可以弥补最近邻法的不足。

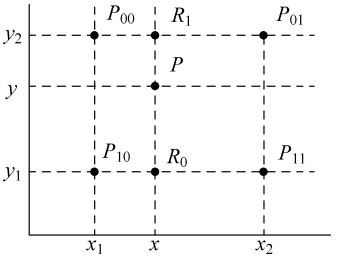


图 5-3　双线性插值法

如图 5-3 所示，已知点 $P_{00}(x_1, y_2)$、$P_{10}(x_1, y_1)$、

$P_{11}(x_2,y_1)$、$P_{01}(x_2,y_2)$，求函数 f 在 $P(x,y)$ 点的值。先在 x 方向线性插值，根据 P_{10}、P_{11}、P_{00}、P_{01} 计算 $f(R_0)$、$f(R_1)$，再在 y 方向线性插值，根据 $f(R_0)$、$f(R_1)$ 计算 $f(P)$。

在 x 方向线性插值：

$$f(R_0)\approx\frac{x_2-x}{x_2-x_1}f(P_{10})+\frac{x-x_1}{x_2-x_1}f(P_{11})\quad \text{当 } R_0=(x,y_1) \tag{5-3}$$

$$f(R_1)\approx\frac{x_2-x}{x_2-x_1}f(P_{00})+\frac{x-x_1}{x_2-x_1}f(P_{01})\quad \text{当 } R_1=(x,y_2) \tag{5-4}$$

在 y 方向线性插值：

$$f(P)\approx\frac{y_2-y}{y_2-y_1}f(R_0)+\frac{y-y_1}{y_2-y_1}f(R_1) \tag{5-5}$$

$$f(P)=f(x,y)\approx\frac{y_2-y}{y_2-y_1}\times\frac{x_2-x}{x_2-x_1}f(P_{10})+\frac{y_2-y}{y_2-y_1}\times\frac{x-x_1}{x_2-x_1}f(P_{11})+\frac{y-y_1}{y_2-y_1}\times\frac{x_2-x}{x_2-x_1}f(P_{00})+\frac{y-y_1}{y_2-y_1}\times\frac{x-x_1}{x_2-x_1}f(P_{01}) \tag{5-6}$$

【例 5-2】 双线性插值缩放图像。

解：

(1) 读取彩色图像。

(2) 双线性插值缩放。

(3) 显示图像，代码如下：

```
#chapter5_2.py
import cv2
import numpy as np
import matplotlib.pyplot as plt

def insert_size(img, nh, nw):
    '''
    双线性插值
    :param img: 彩色图像
    :param nh: 新图像的高度
    :param nw: 新图像的宽度
    :return: 新图像
    '''
    #1. 根据原图、新图尺寸[nh, nw]计算缩放系数
    #原图尺寸
    h, w, c = img.shape
    #新图像
    img_new = np.zeros([nh, nw, c], np.uint8)
    #缩放系数
    h_scale = nh *1.0 / h
    w_scale = nw *1.0 / w
    #2. 对新图每个位置进行线性插值
```

```
    for i in range(nh):
        for j in range(nw):
            #新图坐标[i,j]在原图上对应的非整数作坐标[i_new,j_new]
            i_new = np.round(i / h_scale, 1)                    #例如 i_new = 0.6
            j_new = np.round(j / w_scale, 1)                    #例如 j_new = 0.8
            #把 i_new 和 j_new 转换成字符串,获取坐标整数部分和小数部分
            i_new1, i_new2 = str(i_new).split('.')              #('0', '6')
            j_new1, j_new2 = str(j_new).split('.')              #('0', '8')
            i_new1, i_new2 = int(i_new1), int(i_new2) *0.1      #0, 0.6
            j_new1, j_new2 = int(j_new1), int(j_new2) *0.1      #0, 0.8
            #获取目标坐标[i_new,j_new]周围 4 个整数坐标的像素 img_00、img_10、img_01、
#img_11
            img_00 = img[i_new1, j_new1]                        #img[0,0] 左上角
            #处理越界像素
            if i_new1 + 1 <= h - 1 and j_new1 + 1 <= w - 1:
                img_10 = img[i_new1 + 1, j_new1]                #img[1,0] 左下角
                img_01 = img[i_new1, j_new1 + 1]                #img[0,1] 右上角
                img_11 = img[i_new1 + 1, j_new1 + 1]            #img[1 1] 右下角
                #双线性插值
                r0 = img_10 *(1 - j_new2) + img_11 *j_new2
                r1 = img_00 *(1 - j_new2) + img_01 *j_new2
                r = r1 *(1 - i_new2) + r0 *i_new2
                img_new[i, j] = r
            else:
                img_new[i, j] = img_00
    return img_new

if __name__ == '__main__':
    #1. 读取彩色图像
    img = cv2.imread('pictures/L1.png', 1)
    #2. 双线性插值缩放
    #缩小
    img1 = insert_size(img, nh=40, nw=80)
    #放大
    img2 = insert_size(img, nh=1600, nw=800)
    #3. 显示图像
    plt.subplot(131)
    plt.axis('off')
    plt.imshow(img2[..., ::-1])
    plt.subplot(132)
    plt.axis('off')
    plt.imshow(img[..., ::-1])
    plt.subplot(133)
    plt.axis('off')
    plt.imshow(img1[..., ::-1])
```

运行结果如图 5-4 所示。

(a) 放大　　(b) 原图　　(c) 缩小

图 5-4　双线性插值缩放图像

5.1.2　语法函数

OpenCV 调用函数 cv2.resize()实现图像缩放，其语法格式为

dst＝cv2.resize(src,dsize,fx,fy,interpolation)

(1) dst：输出图像。

(2) src：输入图像。

(3) dsize：输入图像尺寸。

(4) fx,fy：输入图像与输出图像的尺寸比例。dsize 与 fx、fy 设置一项即可。

(5) interpolation：插值方法，见表 5-1。

表 5-1　插值方法

类　　型	注　　释
cv2.INTER_LINEAR	双线性插值
cv2.INTER_NEAREST	最邻近插值
cv2.INTER_CUBIC	三次样条插值
cv2.INTER_AREA	区域插值，根据当前像素周边区域的像素实现当前像素的采样
cv2.INTER_LANCZOS4	一种使用 8×8 近邻的 Lanczos 插值方法
cv2.INTER_LINEAR_EXACT	位精确双线性插值
cv2.INTER_MAX	差值编码掩码
cv2.WARP_FILL_OUTLIERS	标志，填补目标图像中的所有像素。如果它们中的一些对应源图像中的奇异点(离群值)，则将它们设置为 0
cv2.WARP_INVERSE_MAP	标志，逆变换

【例 5-3】　图像堆叠。

解：

(1) 读取彩色图像。

(2) 缩放图像并堆叠。

（3）显示图像，代码如下：

```
#chapter5_3.py
import cv2
import matplotlib.pyplot as plt

#1. 读取彩色图像
img = cv2.imread('pictures/L3.png', 1)
#2. 用 OpenCV 自带函数缩放图像并堆叠
img1 = cv2.resize(img, (120, 120), interpolation=cv2.INTER_LINEAR)
img2 = cv2.resize(img, None, fx=3, fy=3, interpolation=cv2.INTER_NEAREST)
#把缩放后的图像放在一起
img[-120:, -120:] = img1
h, w = img.shape[:2]
img2[-h:, -w:] = img
#3. 显示图像
plt.imshow(img2[..., ::-1])
cv2.imwrite('pictures/p5_3.jpg', img2)
```

运行结果如图 5-5 所示。

图 5-5　OpenCV 自带函数缩放图像并堆叠

5.2 图像翻转

图像翻转是指图像沿轴线进行对称变换，包括上下翻转、左右翻转、上下左右翻转。

5.2.1 基本原理

对图像翻转可以视为对多维数组的操作，如图 5-6 所示，对图像进行上下翻转，相当于对图像的行进行对称变换，第 i 行经过翻转后为(height－i)行。对图像进行左右翻转，相当于对图像的列进行对称变换，第 i 列经过翻转后为(width－i)列。对图像进行上下左右翻转，相当于对图像先进行行对称变换，再对列进行对称变换。

8	7	1
11	6	5
2	1	10

(a) 原图

2	1	10
11	6	5
8	7	1

(b) 上下翻转

1	7	8
5	6	11
10	1	2

(c) 左右翻转

10	1	2
5	6	11
1	7	8

(d) 上下左右翻转

图 5-6 图像翻转

已知原图像素坐标为(x_0,y_0)，对应翻转后图像坐标为(x,y)，原图宽和高为(h,w)。

1. 上下翻转

对图像进行上下翻转操作，(x_0,y_0)与(x,y)的关系如下：

$$\begin{cases} x = x_0 \\ y = h - y_0 \end{cases} \tag{5-7}$$

用矩阵表示：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & h \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{5-8}$$

用代码表示：

```
#通过处理数组实现翻转
h,w = img.shape[:2]
img_new = img.copy()
for j in range(h):
    img_new[j]= img[h-1-j]

#通过仿射实现翻转
mirrorM = np.array([
    [1, 0, 0],
    [0, -1, h]
], dtype=np.float32)
img4 = cv2.warpAffine(img, mirrorM, dsize=img.shape[:2][::-1])
```

2. 左右翻转

对图像进行左右翻转操作，(x_0, y_0)与(x, y)的关系如下：

$$\begin{cases} x = w - x_0 \\ y = y_0 \end{cases} \tag{5-9}$$

用矩阵表示：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & w \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{5-10}$$

用代码表示：

```
#通过处理数组实现翻转
h,w = img.shape[:2]
img_new = img.copy()
for j in range(h):
    img_new[::,j]= img[::,w-1-j]

#通过仿射实现翻转
mirrorM = np.array([
    [-1, 0, w],
    [0, 1, 0]
], dtype=np.float32)
img4 = cv2.warpAffine(img, mirrorM, dsize=img.shape[:2][::-1])
```

3. 上下左右翻转

对图像进行上下左右翻转操作，(x_0, y_0)与(x, y)的关系如下：

$$\begin{cases} x = w - x_0 \\ y = h - y_0 \end{cases} \tag{5-11}$$

用矩阵表示：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & w \\ 0 & -1 & h \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{5-12}$$

用代码表示：

```
h,w = img.shape[:2]
img_new = img.copy()
img_n = img_new.copy()
for j in range(h):
    img_new[j]= img[h-1-j]
for j in range(w):
    img_n[::,j] = img_new[::,w-1-j]

#通过仿射实现翻转
mirrorM = np.array([
    [-1, 0, h],
```

```
    [0, -1, w]
], dtype=np.float32)
img4 = cv2.warpAffine(img, mirrorM, dsize=img.shape[:2][::-1])
```

【例 5-4】 图像翻转源码复现。

解：

(1) 读取图像。

(2) 图像翻转。

(3) 显示图像,代码如下：

```
#chapter5_4.py
import cv2
import numpy as np
import matplotlib.pyplot as plt

def my_flip(img, flip=0):
    '''
    图像翻转
    :param img: 原图
    :param flip: 0(上下翻转),1(左右翻转),-1(上下左右翻转)
    :return: 翻转图像
    '''
    h, w = img.shape[:2]
    img_new = img.copy()
    if flip == 0:
        for j in range(h):
            img_new[j] = img[h - 1 - j]
    elif flip == 1:
        for j in range(w):
            img_new[:, j] = img[:, w - 1 - j]
    elif flip == -1:
        img_n = img_new.copy()
        for j in range(h):
            img_new[j] = img[h - 1 - j]
        for j in range(w):
            img_n[:, j] = img_new[:, w - 1 - j]
        img_new = img_n
    return img_new

if __name__ == '__main__':
    #1. 读取图像
    img = cv2.imread('pictures/L1.png', 1)
    h, w = img.shape[:2]
    #2. 图像翻转
    img1 = my_flip(img, -1)     #上下左右翻转
    img2 = my_flip(img, 0)      #上下翻转
    img3 = my_flip(img, 1)      #左右翻转
    #通过仿射实现镜像
    mirrorM = np.array([
```

```
        [-1, 0, w],
        [0, 1, 0]
    ], dtype=np.float32)
    img4 = cv2.warpAffine(img, mirrorM, dsize=img.shape[:2][::-1])
    #3. 显示图像
    re = np.hstack([img, img1, img2, img3, img4])
    plt.axis('off')
    plt.imshow(re[..., ::-1])
    cv2.imwrite('pictures/p5_7.jpeg', re)
```

运行结果如图 5-7 所示。

(a) 原图　(b) 上下左右翻转　(c) 上下翻转　(d) 左右翻转1　(e) 左右翻转2

图 5-7　图像翻转

5.2.2　语法函数

OpenCV 调用函数 cv2.flip()实现图像的翻转，其语法函数为

dst＝cv2.flip(src,flipCode)

(1) dst：输出图像。

(2) src：原始图像。

(3) flipCode：旋转类型。如果 flipCode 为 0，则表示上下翻转；如果 flipCode 为正数，则表示左右翻转；如果 flipCode 为负数，则表示上下左右翻转。

【例 5-5】 图像翻转。

解：

(1) 读取图像。

(2) 图像翻转。

(3) 显示图像，代码如下：

```
#chapter5_5.py
import cv2
import numpy as np
import matplotlib.pyplot as plt

#1. 读取图像
img = cv2.imread('pictures/L1.png', 1)
#2. 图像翻转
```

```
img1 = cv2.flip(img, -1)      #上下左右翻转
img2 = cv2.flip(img, 0)       #上下翻转
img3 = cv2.flip(img, 1)       #左右翻转
#3. 显示图像
re = np.hstack([img, img1, img2, img3])
plt.imshow(re[..., ::-1])
cv2.imwrite('pictures/p5_8.jpeg', re)
```

运行结果如图 5-8 所示。

(a) 原图　(b) 上下左右翻转　(c) 上下翻转　(d) 左右翻转

图 5-8　图像翻转

5.3　图像平移

图像平移是指将图像中的所有点按照指定的平移量在水平或垂直方向上移动。

5.3.1　基本原理

已知原图像素坐标为(x_0,y_0)，对应翻转后图像坐标为(x,y)，原图宽和高为(h,w)。对图像进行平移，水平方向和垂直方向的平移量为(r,c)，其中(r,c)可以取正值或负值，平移量绝对值不能大于图像的宽和高。(x_0,y_0)与(x,y)的关系如下：

$$\begin{cases} x = x_0 + r \\ y = y_0 + c \end{cases} \tag{5-13}$$

用矩阵表示：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & r \\ 0 & 1 & c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{5-14}$$

用代码表示：

```
def move(img,r,c):
    '''
    :param img:
    :param r: 水平方向平移量
    :param c: 垂直方向平移量
```

```
    :return: img1 平移后的图像
    '''
    h, w = img.shape[:2]
    img1 = np.ones_like(img, np.uint8)
    for i in range(h):
        for j in range(w):
            l1 = i+c
            l2 = j+r
            if l1>=0 and l1<h and l2>=0 and l2<w:
                img1[l1,l2] = img[i,j]
    return img1
```

【例 5-6】 图像平移代码复现。

解：

(1) 读取彩色图像。

(2) 图像平移。

(3) 显示图像,代码如下：

```
#chapter5_6.py
import cv2
import numpy as np
import matplotlib.pyplot as plt

def move(img, r, c):
    '''
    :param img: 原图
    :param r: 水平方向平移量
    :param c: 垂直方向平移量
    :return: img1 平移后的图像
    '''
    h, w = img.shape[:2]
    img1 = np.ones_like(img, np.uint8)
    for i in range(h):
        for j in range(w):
            l1 = i + c
            l2 = j + r
            if l1 >= 0 and l1 < h and l2 >= 0 and l2 < w:
                img1[l1, l2] = img[i, j]
    return img1

if __name__ == '__main__':
    #1. 读取彩色图像
    img = cv2.imread('pictures/L1.png', 1)
    #2. 图像平移
    img1 = move(img, r=80, c=50)
    img2 = move(img, r=80, c=-80)
    img3 = move(img, r=-80, c=50)
    img4 = move(img, r=-80, c=-80)
    #3. 显示图像
    re = np.hstack([img, img1, img2, img3, img4])
    plt.imshow(re[..., ::-1])
    cv2.imwrite('pictures/p5_9.jpeg', re)
```

运行结果如图 5-9 所示。

(a) 原图　(b) 向右向下　(c) 向右向上　(d) 向左向下　(e) 向左向上

图 5-9　图像平移

5.3.2　语法函数

图像平移是仿射变换的一种，OpenCV 调用仿射函数 cv2.warpAffine()实现图像平移，其语法格式为

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

(1) dst：仿射后的输出图像。

(2) src：仿射的原始图像。

(3) M：一个 2×3 的平移矩阵。

(4) dsize：输出图像的尺寸。

(5) flags：表示插值方法，包括线性插值(cv2.INTER_LINEAR)、最近邻插值(cv2.INTER_NEAREST)、三次样条插值(cv2.INTER_CUBIC)、区域插值(cv2.INTER_AREA)。

(6) borderMode：边界像素处理模式(可选项)，包括常量填充(cv2.BORDER_CONSTANT)、复制边界像素(cv2.BORDER_REPLICATE)、反射边界(cv2.BORDER_REFLECT)、包裹边界(cv2.BORDER_WRAP)。

(7) borderValue：边界像素填充值，默认值为 0。

【例 5-7】 图像平移。

解：

(1) 读取彩色图像。

(2) 图像平移。

(3) 显示图像。

```
#chapter5_7.py
import cv2
import numpy as np
import matplotlib.pyplot as plt

#1. 读取彩色图像
img = cv2.imread('pictures/L1.png', 1)
h, w = img.shape[:2]
#2. 图像平移
```

```
#向右向下
m1 = np.float32([[1, 0, 80], [0, 1, 80]])
img1 = cv2.warpAffine(img, m1, (w, h))
#向右向上
m2 = np.float32([[1, 0, 80], [0, 1, -80]])
img2 = cv2.warpAffine(img, m2, (w, h))
#向左向下
m3 = np.float32([[1, 0, -80], [0, 1, 80]])
img3 = cv2.warpAffine(img, m3, (w, h))
#向左向上
m4 = np.float32([[1, 0, -80], [0, 1, -80]])
img4 = cv2.warpAffine(img, m4, (w, h))
#3. 显示图像
re = np.hstack([img1, img2, img3, img4])
plt.imshow(re[..., ::-1])
cv2.imwrite('pictures/p5_10.jpeg', re)
```

运行结果如图 5-10 所示。

(a) 向右向下　(b) 向右向上　(c) 向左向下　(d) 向左向上

图 5-10　图像平移

5.4　图像错切

图像错切(shearing)是对图像在水平或垂直方向按照角度 θ 偏移一定的距离。图像错切分为垂直错切和水平错切，如图 5-11(a)所示，图像 y 轴方向坐标不变，x 轴方向坐标向右平移一定距离。在图 5-11(b)中，图像 x 轴方向坐标不变，y 轴方向坐标向下平移一定距离。

(a) 水平错切　(b) 垂直错切

图 5-11　图像错切

5.4.1 基本原理

已知原图像素坐标为(x_0,y_0),错切后图像坐标为(x,y),原图宽和高为(h,w)。对图像进行水平错切,错切角度为θ,(x_0,y_0)与(x,y)的关系如下:

$$\begin{cases} x = x_0 + \tan\theta \times y_0 \\ y = y_0 \end{cases} \tag{5-15}$$

用矩阵表示:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \tan\theta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{5-16}$$

用代码表示:

```
def h_cut(img,t= 0.8):
    '''
    水平错切
    :param img:彩色图像
    :param theta:角度的正切值 tan(theate)
    :return:水平错切图像
    '''
    h, w ,c = img.shape
    img_n = np.zeros([h, 2*w ,c], dtype=np.uint8)
    for i in range(h):
        for j in range(w):
            #t = tan(theate); x_new = x_0+t *y_0 ;y_new = y_0
            ind_r = int(i+t *j)
            if ind_r < 2*w:
                img_n[i,ind_r] = img[i,j]
    return img_n
```

对图像进行垂直错切,错切角度为θ,(x_0,y_0)与(x,y)的关系如下:

$$\begin{cases} x = x_0 \\ y = y_0 + \tan\theta \cdot x_0 \end{cases} \tag{5-17}$$

用矩阵表示:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \tan\theta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{5-18}$$

用代码表示:

```
def v_cut(img,t = 0.8):
    '''
    垂直错切
    :param img:彩色图像
    :param t:角度的正切值 tan(theate)
    :return:垂直错切图像
```

```
    '''
    h, w ,c = img.shape
    img_n = np.zeros([2*h, w ,c], dtype=np.uint8)
    for i in range(h):
        for j in range(w):
            #t = tan(theate); y_new = y_0+t *x_0 ;x_new = x_0
            ind_c = int(j+t *i)
            if ind_c < 2*h:
                img_n[ind_c,j] = img[i,j]
    return img_n
```

【例 5-8】 图像错切。

解：

(1) 读取彩色图像。

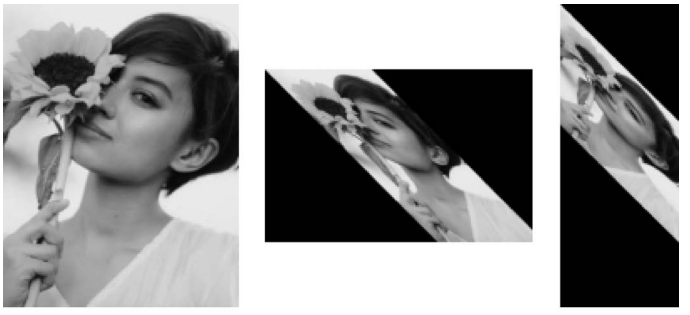
(2) 图像错切。

(3) 显示图像,代码如下：

```
#chapter5_8.py
import cv2
import numpy as np
import matplotlib.pyplot as plt

def h_cut(img, t=0.8):
    '''
    水平错切
    :param img:图像
    :param theta:角度的正切值 tan(theate)
    :return:水平错切
    '''
    h, w, c = img.shape
    img_n = np.zeros([h, 2 *w, c], dtype=np.uint8)
    for i in range(h):
        for j in range(w):
            #t = tan(theate); x_new = x_0+t *y_0 ;y_new = y_0
            ind_r = int(i + t *j)
            if ind_r < 2 *w:
                img_n[i, ind_r] = img[i, j]
    return img_n

def v_cut(img, t=0.8):
    '''
    垂直错切
    :param img:图像
    :param theta:角度的正切值 tan(theate)
    :return:垂直错切图像
    '''
    h, w, c = img.shape
    img_n = np.zeros([2 *h, w, c], dtype=np.uint8)
    for i in range(h):
```

```
            for j in range(w):
                #t = tan(theate); y_new = y_0+t*x_0 ;x_new = x_0
                ind_c = int(j + t*i)
                if ind_c < 2*h:
                    img_n[ind_c, j] = img[i, j]
        return img_n

    if __name__ == '__main__':
        #1. 读取彩色图像
        img = cv2.imread('pictures/L1.png', 1)
        #2. 图像错切
        #水平错切
        img1 = h_cut(img, t=0.8)
        #垂直错切
        img2 = v_cut(img, t=0.8)
        #3. 显示图像
        plt.subplot(131)
        plt.axis('off')
        plt.imshow(img[..., ::-1])
        plt.subplot(132)
        plt.axis('off')
        plt.imshow(img1[..., ::-1])
        plt.subplot(133)
        plt.axis('off')
        plt.imshow(img2[..., ::-1])
```

运行结果如图 5-12 所示。



(a) 原图　　(b) 水平错切　　(c) 垂直错切

图 5-12　图像错切

5.4.2　语法函数

图像错切是仿射变换的一种，OpenCV 调用仿射函数 cv2.warpAffine()实现图像错切，参考 5.3.2 节该函数的语法格式。

【例 5-9】 图像错切。

解：

(1) 读取彩色图像。

（2）图像错切。设置错切矩阵，根据错切矩阵对原图进行仿射变换。

（3）显示图像，代码如下：

```
#chapter5_9.py
import cv2
import numpy as np
import matplotlib.pyplot as plt

#1. 读取彩色图像
img = cv2.imread('pictures/L1.png', 1)
#2. 图像错切
tan0 = 0.5
#水平错切
#水平错切矩阵
transM_h = np.array([
    [1, tan0, 0],
    [0, 1, 0]
], dtype=np.float32)
#仿射变换
img_trans_h = cv2.warpAffine(img, transM_h, dsize=(600, 600))
#垂直错切
#垂直错切矩阵
transM_v = np.array([
    [1, 0, 0],
    [tan0, 1, 0]
], dtype=np.float32)
#仿射变换
img_trans_v = cv2.warpAffine(img, transM_v, dsize=(600, 600))  #transM:平移矩阵;
#dsize:平移后图像的新的宽和高
#3. 显示图像
re = np.hstack([img_trans_h, img_trans_v])
plt.imshow(re[..., ::-1])
cv2.imwrite('pictures/p5_13.jpeg', re)
```

运行结果如图 5-13 所示。

(a) 水平错切　　(b) 垂直错切

图 5-13　图像错切

5.5 图像旋转

图像旋转是指图像沿着任意点，按顺时针方向把图像的所有像素旋转 θ 角度。

5.5.1 基本原理

如图 5-14 所示，点 P_0 是圆心为 O 半径为 r 的圆上的一点，点 $P_0(x_0,y_0)$ 与水平方向的夹角为 α，点 P_0 按顺时针方向旋转 β 角度后得到 $P(x,y)$。

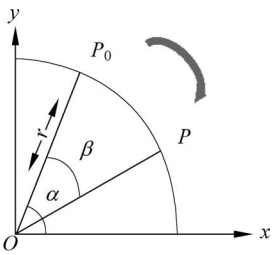


图 5-14 坐标旋转

点 $P_0(x_0,y_0)$ 用极坐标表示：

$$\begin{cases} x_0 = r \times \cos\alpha \\ y_0 = r \times \sin\alpha \end{cases} \tag{5-19}$$

点 $P(x,y)$ 用极坐标表示：

$$\begin{cases} x = r \times \cos(\alpha-\beta) = r \times \cos\alpha \times \cos\beta + r \times \sin\alpha \times \sin\beta \\ y = r \times \sin(\alpha-\beta) = r \times \sin\alpha \times \cos\beta - r \times \cos\alpha \times \sin\beta \end{cases} \tag{5-20}$$

化简得

$$\begin{cases} x = x_0 \times \cos\beta + y_0 \times \sin\beta \\ y = -x_0 \times \sin\beta + y_0 \times \cos\beta \end{cases} \tag{5-21}$$

用矩阵表示：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{5-22}$$

旋转矩阵 $\boldsymbol{R}$ 为

$$\boldsymbol{R} = \begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-23}$$

上述坐标旋转是以图像原点为中心的，如果指定某点作为旋转中心，则图像先以原点（左上角）为中心旋转，再把旋转后的某点平移到旋转前的位置。例如，如果以 $C_0(x_0,y_0)$ 为旋转中心，则旋转后的坐标为 $C(x,y)$，旋转平移量为 $\boldsymbol{CC}_0 = C_0 - C$。用矩阵表示：

$$\boldsymbol{CC}_0 = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} x_0 - x \\ y_0 - y \end{bmatrix} = \begin{bmatrix} x_0 \times (1-\cos\beta) - y_0 \times \sin\beta \\ x_0 \times \sin\beta + y_0 \times (1-\cos\beta) \end{bmatrix} \tag{5-24}$$

则平移矩阵 $\boldsymbol{T}$ 为

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \tag{5-25}$$

坐标经旋转和平移，对应的矩阵 $\boldsymbol{M}$ 为

$$\boldsymbol{M}=\boldsymbol{TR}=\begin{bmatrix}1 & 0 & \Delta x\\ 0 & 1 & \Delta y\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\beta & \sin\beta & 0\\ -\sin\beta & \cos\beta & 0\\ 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}\cos\beta & \sin\beta & \Delta x\\ -\sin\beta & \cos\beta & \Delta y\\ 0 & 0 & 1\end{bmatrix}$$
$$=\begin{bmatrix}\cos\beta & \sin\beta & x_0\times(1-\cos\beta)-y_0\times\sin\beta\\ -\sin\beta & \cos\beta & x_0\times\sin\beta+y_0\times(1-\cos\beta)\\ 0 & 0 & 1\end{bmatrix} \tag{5-26}$$

以图像原点为中心的旋转代码：

```
def rote(img,theta):
    '''
    旋转
    :param img:输入图像
    :param theta:旋转角度
    :return:旋转后的图像
    '''
    h, w = img.shape[:2]
    #存放旋转后的图像
    img1 = np.zeros_like(img, np.uint8)
    #把角度转换成弧度
    t = theta/180*np.pi
    for i in range(h):
        for j in range(w):
            #r 为旋转后图像的行坐标,c 为旋转后图像的列坐标
            r = int(-np.sin(t)*j+np.cos(t)*i)
            c = int(np.cos(t)*j+np.sin(t)*i)
            if r>=0 and r+1<h and c >=0 and c+1<w:
                img1[r,c] = img1[r+1,c+1] = img[i,j]
    return img1
```

以图像任意点为中心进行旋转，代码如下：

```
def rote_center(img,theta,x0,y0):
    '''
    以任意点为旋转中心
    :param img: 输入图像
    :param theta:旋转角度
    :param x0:旋转中心的横坐标
    :param y0:旋转中心的纵坐标
    :return:旋转后的图像
    '''
    h, w = img.shape[:2]
    #存放旋转后的图像
    img1 = np.zeros_like(img, np.uint8)
    #把角度转换成弧度
    t = theta/180*np.pi
    for i in range(h):
        for j in range(w):
            #r 为旋转后图像的行坐标,c 为旋转后图像的列坐标
```

```
            r = int(-np.sin(t)*j+np.cos(t)*i+ x0*np.sin(t) +y0*(1-np.cos(t)))
            c = int(np.cos(t)*j+np.sin(t)*i+x0*(1- np.cos(t)) - y0 *np.sin(t))
            if r>=0 and r+1<h and c>=0 and c+1<h:
                img1[r,c] = img1[r+1,c+1] = img[i,j]
    return img1
```

5.5.2 语法函数

当OpenCV调用函数cv2.warpAffine()对图像进行旋转时，先通过函数cv2.getRotationMatrix2D()获取旋转矩阵，其语法格式为

retval＝cv2.getRotationMatrix2D(center,angle,scale)

(1) center：旋转的中心点。

(2) angle：旋转角度，正数表示逆时针旋转，负数表示顺时针旋转。

(3) scale：变换尺度(缩放大小)。

【例5-10】 图像旋转。

解：

(1) 读取彩色图像。

(2) 图像旋转。设置旋转矩阵，根据旋转矩阵对原图进行仿射变换。

(3) 显示图像，代码如下：

```
#chapter5_10.py
import cv2
import numpy as np
import matplotlib.pyplot as plt

def rote(img, theta):
    '''
    旋转
    :param img:输入图像
    :param theta:旋转角度
    :return:旋转后的图像
    '''
    h, w = img.shape[:2]
    #存放旋转后的图像
    img1 = np.zeros_like(img, np.uint8)
    #把角度转换成弧度
    t = theta / 180 *np.pi
    for i in range(h):
        for j in range(w):
            #r为旋转后图像的行坐标,c为旋转后图像的列坐标
            r = int(-np.sin(t) *j + np.cos(t) *i)
            c = int(np.cos(t) *j + np.sin(t) *i)
            if r >= 0 and r + 1 < h and c >= 0 and c + 1 < w:
                img1[r, c] = img1[r + 1, c + 1] = img[i, j]
    return img1

def rote_center(img, theta, x0, y0):
```

```
    '''
    以任意点为旋转中心
    :param img: 输入图像
    :param theta:旋转角度
    :param x0:旋转中心的横坐标
    :param y0:旋转中心的纵坐标
    :return:旋转后的图像
    '''
    h, w = img.shape[:2]
    #存放旋转后的图像
    img1 = np.zeros_like(img, np.uint8)
    #把角度转换成弧度
    t = theta / 180 *np.pi
    for i in range(h):
        for j in range(w):
            #r 为旋转后图像的行坐标,c 为旋转后图像的列坐标
            r = int(-np.sin(t) *j + np.cos(t) *i + x0 *np.sin(t) + y0 *(1 - np.cos(t)))
            c = int(np.cos(t) *j + np.sin(t) *i + x0 *(1 - np.cos(t)) - y0 *np.sin(t))
            if r >= 0 and r + 1 < h and c >= 0 and c + 1 < w:
                img1[r, c] = img1[r + 1, c + 1] = img[i, j]
    return img1

if __name__ == '__main__':
    #1. 读取彩色图像
    img = cv2.imread('pictures/L1.png', 1)
    h, w = img.shape[:2]
    #2. 图像旋转
    #以原点为旋转中心
    img1 = rote(img, theta=30)
    #以任意点为旋转中心
    img2 = rote_center(img, theta=30, x0=h //2, y0=w //2)
    #OpenCV 自带函数旋转
    rotateM = cv2.getRotationMatrix2D((w //2, h //2), 30, 1)
    img_rotate = cv2.warpAffine(img, rotateM, dsize=(w, h))
    #3. 显示图像
    re = np.hstack([img, img1, img2, img_rotate])
    plt.imshow(re[..., ::-1])
    cv2.imwrite('pictures/p5_15.jpeg', re)
```

运行结果如图 5-15 所示。

(a) 原图　(b) 原点旋转　(c) 中心旋转1　(d) 中心旋转2

图 5-15　图像旋转

如图 5-15 所示，图 5-15(a)为原图，图 5-15(b)为以图像左上角为中心，旋转 30°后的图像，图 5-15(c)和图 5-15(d)均为以图像中心为中心，旋转 30°后的图像。

5.6 仿射变换

仿射变换(Affine Transform)指一个向量空间进行一次线性变换并接上一个平移，变成另一个向量空间。图像中原来的直线、平行线经过仿射变换后还是直线、平行线。仿射变换前一条直线上两条线段的比例，在变换后比例不变。

5.6.1 基本原理

仿射变换可以分解为缩放、翻转、旋转和错切的组合。已知原图像素坐标为(x_0,y_0)，仿射变换后图像坐标为(x,y)，s_x、s_y 为缩放系数，φ、θ 为错切角度，β 为旋转角度，r、c 为宽和高平移数据，$\boldsymbol{A}$ 为仿射变换矩阵。(x_0,y_0)与(x,y)的关系如下：

$$\begin{bmatrix}x\\y\\1\end{bmatrix}=\begin{bmatrix}s_x&0&0\\0&s_y&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&\tan\varphi&0\\\tan\theta&1&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}\cos\beta&\sin\beta&0\\-\sin\beta&\cos\beta&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&0&r\\0&1&c\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}x_0\\y_0\\1\end{bmatrix}\tag{5-27}$$

其中，

$$\boldsymbol{A}=\begin{bmatrix}s_x&0&0\\0&s_y&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&\tan\varphi&0\\\tan\theta&1&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}\cos\beta&\sin\beta&0\\-\sin\beta&\cos\beta&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&0&r\\0&1&c\\0&0&1\end{bmatrix}=\begin{bmatrix}a_1&a_2&a_3\\a_4&a_5&a_6\\0&0&1\end{bmatrix}\tag{5-28}$$

仿射变换化简后可得

$$\begin{bmatrix}x\\y\\1\end{bmatrix}=\begin{bmatrix}a_1&a_2&a_3\\a_4&a_5&a_6\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}x_0\\y_0\\1\end{bmatrix}=\boldsymbol{A}\cdot\begin{bmatrix}x_0\\y_0\\1\end{bmatrix}\tag{5-29}$$

其中，$\boldsymbol{A}$ 是线性变换(缩放、旋转、错切)和平移变换的叠加。仿射变换需先求出变换矩阵 $\boldsymbol{A}$。变换矩阵 $\boldsymbol{A}$ 中有 6 个未知参数，因此需要至少 3 对不共线的点来求解未知参数。已知原图上的 3 个不共线的点(x_0,y_0)、(x_1,y_1)、(x_2,y_2)和仿射变换后对应的点(x_{00},y_{00})、(x_{11},y_{11})、(x_{22},y_{22})。

用 3 对点写出 6 个方程组：

$$\begin{aligned}a_1\cdot x_0+a_2\cdot y_0+a_3&=x_{00}\\a_4\cdot x_0+a_5\cdot y_0+a_6&=y_{00}\\a_1\cdot x_1+a_2\cdot y_1+a_3&=x_{11}\\a_4\cdot x_1+a_5\cdot y_1+a_6&=y_{11}\end{aligned}$$

$$a_1 \cdot x_2 + a_2 \cdot y_2 + a_3 = x_{22}$$
$$a_4 \cdot x_2 + a_5 \cdot y_2 + a_6 = y_{22} \tag{5-30}$$

仿射变换矩阵用 $\boldsymbol{X}$ 表示，系数矩阵用 $\boldsymbol{M}$ 表示，变换后的坐标用向量 $\boldsymbol{B}$ 表示：

$$\boldsymbol{X} = [a_1, a_2, a_3, a_4, a_5, a_6]^{\mathrm{T}} \tag{5-31}$$

$$\boldsymbol{B} = [x_{00}, y_{00}, x_{11}, y_{11}, x_{22}, y_{22}]^{\mathrm{T}} \tag{5-32}$$

$$\boldsymbol{M} = \begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 \\ x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \end{bmatrix} \tag{5-33}$$

$$\boldsymbol{M} \cdot \boldsymbol{X} = \boldsymbol{B} \tag{5-34}$$

直接对 $\boldsymbol{M}$ 求逆，可以求出仿射变换矩阵：

$$\boldsymbol{X} = \boldsymbol{M}^{-1}\boldsymbol{B} \tag{5-35}$$

仿射变换的步骤如下：

(1) 获取仿射变换矩阵。根据原图上的 3 个不共线的点(x_0, y_0)、(x_1, y_1)、(x_2, y_2)和仿射变换后对应的点(x_{00}, y_{00})、(x_{11}, y_{11})、(x_{22}, y_{22})获取数矩阵 $\boldsymbol{M}$、变换后的坐标 $\boldsymbol{B}$，根据 $\boldsymbol{X} = \boldsymbol{M}^{-1}\boldsymbol{B}$ 求得参数矩阵 $\boldsymbol{X}$，把仿射变换矩阵 $\boldsymbol{X}$ 变成仿射变换矩阵 $\boldsymbol{A}$。

(2) 仿射变换。将仿射变换矩阵的逆矩阵与目标图像的坐标相乘，便可得到对应原图的坐标，然后把目标图像坐标的像素用原图对应坐标的像素填充。如果想得到更好的变换效果，则可以用双线性插值法获取坐标的像素，代码如下：

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

def get_affine_map(src_points,dst_points):
    '''
    获取仿射变换矩阵
    :param src_points: 数组类型,原图上 3 个不共线的点。shape:(3, 2)
    :param dst_points: 数组类型,原图上 3 个不共线的点透视后对应的 3 个点。shape:(3, 2)
    :return: 仿射变换矩阵
    MX = B;X 为仿射变换参数,B 为目标图像坐标,M 为系数矩阵
    '''
    #变换后的坐标 B
    B = dst_points.flatten().reshape([6,1])
    #系数矩阵 M
    M = np.zeros([6,6])
    for i in range(3):
        p1 = src_points[i]
        M[2*i] = [p1[0],p1[1],1,0,0,0]
```

```
            M[2*i+1] = [0,0,0,p1[0],p1[1],1]
        M = np.mat(M)
        #X 为仿射变换矩阵
        X = M.I *B #X.shape:(6,)
        X = X.reshape([2, 3])
        return np.vstack([X,[0,0,1]]) #[2, 3]-->[3,3]

    def affine_transform(img,X):
        '''
        仿射变换,根据目标图像坐标得到其在原图上的坐标,根据坐标把原图像素赋值给目标图像
        out_idx = X *img_idx --> img_idx = (X.I) *out_idx
        X 为仿射变换矩阵,X.I 为 X 的逆矩阵,img_idx 为原图像坐标,out_idx 为目标图像坐标
        :param img: 原图
        :param X: 仿射变换矩阵
        :return: 经过仿射变换后的图
        '''
        #仿射变换矩阵的逆矩阵
        X_IV = X.I
        h, w = img.shape[:2]
        #out 为目标图像
        out = np.zeros_like(img, np.uint8)
        #遍历目标图像中的每个像素,找到在原图像上对应的位置
        for i in range(h):
            for j in range(w):
                #目标图像坐标
                idx = np.array([j,i,1]).reshape(3,1)
                #原图对应的坐标,p.shape:(3, 1)
                p = np.array(X_IV *idx)
                img_x = int(p[0][0])
                img_y = int(p[1][0])
                #判断原图像坐标是否越界
                if img_y >= 0 and img_y + 1 < h and img_x >= 0 and img_x + 1 < w:
                    out[i,j] = img[img_y,img_x]
        return out
```

5.6.2 语法函数

当 OpenCV 调用函数 cv2.warpAffine()对图像进行仿射时,先通过函数 cv2.getAffineTransform()获取仿射变换矩阵,其语法格式为

retval=cv2.getAffineTransform(src,dst)

(1) retval:仿射变换矩阵。

(2) src:输入图像的 3 个点坐标。

(3) dst:输出图像的 3 个坐标。

【例 5-11】 图像仿射变换。

解:

(1) 读取彩色图像。

(2) 仿射变换。分别用 OpenCV、复现代码实现仿射变换。

（3）显示图像，代码如下：

```
#chapter5_11.py 仿射变换
import cv2
import numpy as np
import matplotlib.pyplot as plt

def get_affine_map(src_points, dst_points):
    '''
    获取仿射变换矩阵
    :param src_points: 数组类型，原图上 3 个不共线的点。shape:(3, 2)
    :param dst_points: 数组类型，原图上 3 个不共线的点透视后对应的 3 个点。shape:(3, 2)
    :return: 仿射变换矩阵
    MX = B;X 为仿射变换参数，B 为目标图像坐标，M 为系数矩阵
    '''
    #变换后的坐标 B
    B = dst_points.flatten().reshape([6, 1])
    #系数矩阵 M
    M = np.zeros([6, 6])
    for i in range(3):
        p1 = src_points[i]
        M[2 *i] = [p1[0], p1[1], 1, 0, 0, 0]
        M[2 *i + 1] = [0, 0, 0, p1[0], p1[1], 1]
    M = np.mat(M)
    #X 为仿射变换矩阵
    X = M.I *B #X.shape:(6,)
    X = X.reshape([2, 3])
    return np.vstack([X, [0, 0, 1]]) #[3,3]

def affine_transform(img, X):
    '''
    仿射变换 MX=B --> M=(X.I)B 目标图像的坐标对应原图上的坐标
    X 为仿射变换参数，X.I 为 X 的逆矩阵，B 为目标图像坐标，M 为系数矩阵
    :param img: 原图
    :param X: 仿射变换矩阵
    :return: 经过仿射变换后的图
    X.I *np.array([50,50,1]).T
    '''
    #仿射变换矩阵的逆矩阵
    X_IV = X.I
    h, w = img.shape[:2]
    #out 为目标图像
    out = np.zeros_like(img, np.uint8)
    #遍历目标图像中的每个像素，找到在原图像上对应的位置
    for i in range(h):
        for j in range(w):
            #目标图像坐标
            idx = np.array([j, i, 1]).reshape(3, 1)
            #原图对应的坐标，p.shape:(3, 1)
            p = np.array(X_IV *idx)
            img_x = int(p[0][0])
            img_y = int(p[1][0])
            #判断原图像坐标是否越界
```

```
                if img_y >= 0 and img_y + 1 < h and img_x >= 0 and img_x + 1 < w:
                    out[i, j] = img[img_y, img_x]
        return out

    if __name__ == '__main__':
        #1. 读取彩色图像
        img = cv2.imread('pictures/L1.png', 1)
        #2. 仿射变换
        heigh, width, _ = img.shape
        #2.1 原图不共线的 3 个点 matSrc,仿射变换后对应的点 matDst
        src_points = np.float32([[0, 0], [0, heigh - 1], [width - 1, 0]])
    #左上角、左下角及右下角坐标
        dst_points = np.float32([[30, 30], [100, heigh - 20], [width - 30, 100]])
    #左上角、左下角及右下角坐标
        #2.2.1 调用 OpenCV 实现仿射变换
        M = cv2.getAffineTransform(src_points, dst_points)
        dst = cv2.warpAffine(img, M, (width, heigh))
        #2.2.2 仿射变换代码复现
        X = get_affine_map(src_points, dst_points)
        out = affine_transform(img, X)
        #3. 显示图像
        re = np.hstack([img, dst, out])
        plt.imshow(re[..., ::-1])
        cv2.imwrite('pictures/p5_16.jpeg', re)
        print(f'M:\n{M}')
        print(f'X:\n{X}')
        #运行结果
        '''
        M:  #代码复现求解的仿射变换矩阵
        [[ 0.82898551  0.15695067 30.  ]
         [ 0.20289855  0.89013453 30.  ]]
        X:  #OpenCV 自带函数求解的仿射变换矩阵
        [[ 0.82898551  0.15695067 30.  ]
         [ 0.20289855  0.89013453 30.  ]
         [ 0.          0.         1.        ]]
        '''
```

运行结果如图 5-16 所示。

(a) 原图　(b) OpenCV自带函数　(c) 代码复现结果

图 5-16　图像仿射变换

5.7　透视变换

透视变换(Perspective Transformation)把空间三维立体投射到投影面上而得到二维平面的过程。具体做法是先将二维的图像投影到一个三维视平面上,再转换到二维坐标,因此也称为投影映射(Projective Mapping)。

5.7.1　基本原理

原图像素坐标$(x_{_0},y_{_0})$从二维空间变换到三维空间(x,y,z),(x,y,z)为透视后的3个坐标,(x',y')为原图像素透视后的二维表示,$\boldsymbol{A}$为透视变换矩阵。透视变换公式表示:

$$\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}a_1&a_2&a_3\\a_4&a_5&a_6\\a_7&a_8&1\end{bmatrix}\begin{bmatrix}x_{\_0}\\y_{\_0}\\1\end{bmatrix}=\boldsymbol{A}\begin{bmatrix}x_{\_0}\\y_{\_0}\\1\end{bmatrix}\tag{5-36}$$

x、y再通过除以z转换成二维坐标。

$$\begin{cases}x'=\dfrac{x}{z}=\dfrac{a_1x_{\_0}+a_2y_{\_0}+a_3}{a_7x_{\_0}+a_8y_{\_0}+1}\\[2ex]y'=\dfrac{y}{z}=\dfrac{a_4x_{\_0}+a_5y_{\_0}+a_6}{a_7x_{\_0}+a_8y_{\_0}+1}\end{cases}\tag{5-37}$$

展开上式:

$$a_1\cdot x_{\_0}+a_2\cdot y_{\_0}+a_3-a_7\cdot x_{\_0}x'-a_8\cdot y\cdot x'=x'\tag{5-38}$$

$$a_4\cdot x_{\_0}+a_5\cdot y_{\_0}+a_6-a_7x_{\_0}\cdot y'-a_8\cdot y_{\_0}\cdot y'=y'\tag{5-39}$$

已知原图上的4个点(x_0,y_0)、(x_1,y_1)、(x_2,y_2)、(x_3,y_3)和透视后图像上的4个(x_{00},y_{00})、(x_{11},y_{11})、(x_{22},y_{22})、(x_{33},y_{33}),代入线性方程组,得到8个方程,求解8个未知数,用等式表示:

$$\begin{aligned}
&a_1\cdot x_0+a_2\cdot y_0+a_3-a_7\cdot x_0\cdot x_{00}-a_8\cdot y_0\cdot x_{00}=x_{00}\\
&a_4\cdot x_0+a_5\cdot y_0+a_6-a_7\cdot x_0\cdot y_{00}-a_8\cdot y_0\cdot y_{00}=y_{00}\\
&a_1\cdot x_1+a_2\cdot y_1+a_3-a_7\cdot x_1\cdot x_{11}-a_8\cdot y_1\cdot x_{11}=x_{11}\\
&a_4\cdot x_1+a_5\cdot y_1+a_6-a_7\cdot x_1\cdot y_{11}-a_8\cdot y_1\cdot y_{11}=y_{11}\\
&a_1\cdot x_2+a_2\cdot y_2+a_3-a_7\cdot x_2\cdot x_{22}-a_8\cdot y_2\cdot x_{22}=x_{22}\\
&a_4\cdot x_2+a_5\cdot y_2+a_6-a_7\cdot x_2\cdot y_{22}-a_8\cdot y_2\cdot y_{22}=y_{22}\\
&a_1\cdot x_3+a_2\cdot y_3+a_3-a_7\cdot x_3\cdot x_{33}-a_8\cdot y_3\cdot x_{33}=x_{33}\\
&a_4\cdot x_3+a_5\cdot y_3+a_6-a_7\cdot x_3\cdot y_{33}-a_8\cdot y_3\cdot y_{33}=y_{33}
\end{aligned}\tag{5-40}$$

透视变换矩阵用$\boldsymbol{X}$表示,系数矩阵用$\boldsymbol{M}$表示,变换后的坐标用$\boldsymbol{B}$表示:

$$\boldsymbol{X}=[a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8]^{\mathrm{T}}\tag{5-41}$$

$$\boldsymbol{B}=[x_{00},y_{00},x_{11},y_{11},x_{22},y_{22},x_{33},y_{33}]^{\mathrm{T}} \tag{5-42}$$

$$\boldsymbol{M}=\begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -x_0\cdot x_{00} & -y_0\cdot x_{00} \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -x_0\cdot x_{00} & -y_0\cdot y_{00} \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1\cdot x_{11} & -y_1\cdot x_{11} \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1\cdot x_{11} & -y_1\cdot x_{11} \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2\cdot x_{22} & -y_2\cdot x_{22} \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2\cdot x_{22} & -y_2\cdot x_{22} \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3\cdot x_{33} & -y_3\cdot x_{33} \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3\cdot x_{33} & -y_3\cdot y_{33} \end{bmatrix} \tag{5-43}$$

$$\boldsymbol{M}\cdot\boldsymbol{X}=\boldsymbol{B} \tag{5-44}$$

直接对 $\boldsymbol{M}$ 求逆，可以求出透视变换矩阵：

$$\boldsymbol{X}=\boldsymbol{M}^{-1}\boldsymbol{B} \tag{5-45}$$

透视变换的步骤如下：

（1）获取透视变换矩阵。根据原图上的 4 个不共线的点(x_0,y_0)、(x_1,y_1)、(x_2,y_2)、(x_3,y_3)和透视变换后对应的点(x_{00},y_{00})、(x_{11},y_{11})、(x_{22},y_{22})、(x_{33},y_{33})获取系数矩阵 $\boldsymbol{M}$、变换后的坐标 $\boldsymbol{B}$，根据 $\boldsymbol{X}=\boldsymbol{M}^{-1}\boldsymbol{B}$ 求得矩阵 $\boldsymbol{X}$，把矩阵 $\boldsymbol{X}$ 变成透视变换矩阵 $\boldsymbol{A}$。

（2）透视变换。透视变换矩阵的逆矩阵与目标图像的坐标相乘而得到对应原图的坐标，把目标图像坐标的像素用原图对应坐标的像素填充，代码如下：

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

def get_projective_map(src_points,dst_points):
    '''
    获取透视变换矩阵
    :param src_points: 数组类型,原图上 4 个不共线的点。shape:(4, 2)
    :param dst_points: 数组类型,原图上 4 个不共线的点透视后对应的 4 个点。shape:(4, 2)
    :return: 透视变换矩阵
    '''
    #变换后的坐标 B
    B = dst_points.flatten().reshape([8,1])
    #系数矩阵 M
    M = np.zeros([8,8])
    for i in range(4):
        p1 = src_points[i]
        p2 = dst_points[i]
        M[2*i] = [p1[0],p1[1],1,0,0,0,-p1[0]*p2[0],-p1[1]*p2[0]]
        M[2*i+1] = [0,0,0,p1[0],p1[1],1,-p1[0]*p2[1],-p1[1]*p2[1]]
    M = np.mat(M)
    #X 为透视变换矩阵
    X = M.I *B #X.shape:(8,)
```

```
        #在 X 末尾添加 1
        X = np.vstack([X, [1]]).reshape([3,3]) #X.shape:(3,3)
        return X

    def projective_transform(img,X):
        '''
        透视变换 out_idx = X * img_idx --> img_idx = (X.I) * out_idx
        X 为透视变换矩阵,X.I 为 X 的逆矩阵,img_idx 为原图像坐标,out_idx 为目标图像坐标
        :param img: 原图
        :param X: 透视变换矩阵
        :return: 目标图像
        '''
        #透视变换矩阵的逆矩阵
        X_IV = X.I
        h, w = img.shape[:2]
        #out 为目标图像
        out = np.zeros_like(img, np.uint8)
        #遍历目标图像中的每个像素,找到在原图像上对应的位置
        for i in range(h):
            for j in range(w):
                #目标图像坐标
                idx = np.array([j,i,1]).reshape(3,1)
                #原图对应的坐标,p.shape:(3, 1)
                p = np.array(X_IV * idx)
                img_x = int(p[0][0])
                img_y = int(p[1][0])
                #判断原图像坐标是否越界
                if img_y >= 0 and img_y + 1 < h and img_x >= 0 and img_x + 1 < w:
                    out[i,j] = img[img_y,img_x]
        return out
```

5.7.2　语法函数

OpenCV 先调用函数 cv2.getPerspectiveTransform()获取透视变换矩阵,再调用函数 cv2.warpPerspective()实现透视变换。获取透视变换矩阵的语法格式为

retval=cv2.getPerspectiveTransform(src,dst)

(1) retval：透视变换矩阵。

(2) src：输入图像的 4 个顶点的坐标。

(3) dst：输出图像的 4 个顶点的坐标。

透视转换的语法格式为

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

(1) dst：透视变换后的图像。

(2) src：原图。

(3) M：透视变换矩阵。

(4) dsize：图像的尺寸。

(5) flags：插值方法。

（6）borderMode：边界填充类型。

（7）borderValue：边界值。

【例 5-12】 透视变换。

解：

（1）读取彩色图像。

（2）透视变换。分别用 OpenCV、复现代码实现透视变换。

（3）显示图像，代码如下：

```
#chapter5_12.py 透视变换
import cv2
import numpy as np
import matplotlib.pyplot as plt

def get_projective_map(src_points, dst_points):
    '''
    获取透视变换矩阵
    :param src_points: 数组类型，原图上 4 个不共线的点。shape:(4, 2)
    :param dst_points: 数组类型，原图上 4 个不共线点透射后对应的 4 个点。shape:(4, 2)
    :return: 透视变换矩阵
    '''
    #变换后的坐标 B
    B = dst_points.flatten().reshape([8, 1])
    #系数矩阵 M
    M = np.zeros([8, 8])
    for i in range(4):
        p1 = src_points[i]
        p2 = dst_points[i]
        M[2 *i] = [p1[0], p1[1], 1, 0, 0, 0, -p1[0] *p2[0], -p1[1] *p2[0]]
        M[2 *i + 1] = [0, 0, 0, p1[0], p1[1], 1, -p1[0] *p2[1], -p1[1] *p2[1]]
    M = np.mat(M)
    #X 为透视变换矩阵
    X = M.I *B #X.shape:(8,)
    #在 X 末尾添加 1
    X = np.vstack([X, [1]]).reshape([3, 3]) #X.shape:(3,3)
    return X

def projective_transform(img, X):
    '''
    透视变换 out_idx = X *img_idx --> img_idx = (X.I)*out_idx
    X 为透视变换矩阵，X.I 为 X 的逆矩阵，img_idx 为原图像坐标，out_idx 为目标图像坐标
    :param img: 原图
    :param X: 透视变换矩阵
    :return: 目标图像
    '''
    #透视变换矩阵的逆矩阵
    X_IV = X.I
    h, w = img.shape[:2]
    #out 为目标图像
    out = np.zeros_like(img, np.uint8)
    #遍历目标图像中的每个像素，找到在原图像上对应的位置
```

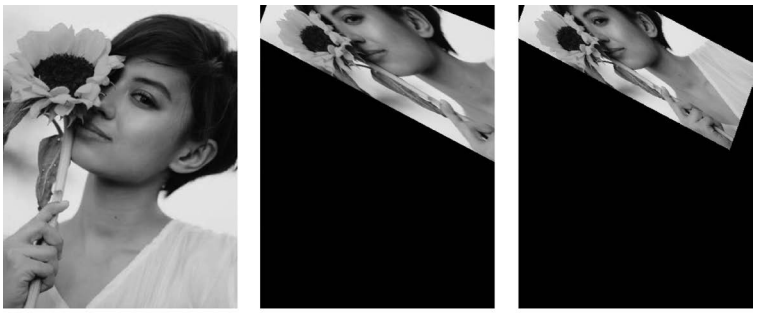
```
    for i in range(h):
        for j in range(w):
            #目标图像坐标
            idx = np.array([j, i, 1]).reshape(3, 1)
            #原图对应的坐标,p.shape:(3, 1)
            p = np.array(X_IV *idx)
            img_x = int(p[0][0])
            img_y = int(p[1][0])
            #判断原图像坐标是否越界
            if img_y >= 0 and img_y + 1 < h and img_x >= 0 and img_x + 1 < w:
                out[i, j] = img[img_y, img_x]
    return out

if __name__ == '__main__':
    #1. 读取彩色图像
    img = cv2.imread('pictures/L1.png')
    #2. 获取透视变换矩阵
    heigh, width, _ = img.shape
    #2.1 原图不共线的 4 个点 matSrc,仿射变换后对应的点 matDst
    src_points = np.float32([[95, 26], [174, 105], [87, 207], [8, 140]])
#左上角、左下角、右下角、右上角坐标
    dst_points = np.float32([[25, 20], [100, 20], [180, 100], [105, 100]])
#左上角、左下角,右下角、右上角坐标
    #2.2.1 调用 OpenCV 实现透视变换矩阵
    M = cv2.getPerspectiveTransform(src_points, dst_points)
    dst = cv2.warpPerspective(img, M, (width, heigh))
    #2.2.2 透视变换代码复现
    X = get_projective_map(src_points, dst_points)
    out = projective_transform(img, X)
    #3. 显示图像
    re = np.hstack([img, dst, out])
    plt.imshow(re[..., ::-1])
    cv2.imwrite('pictures/p5_17.jpeg', re)
    print(f'M:\n{M}')
    print(f'X:\n{X}')
    #运行结果
    '''
    M:  #OpenCV 自带函数求解的透视变换矩阵
    [[ 1.27430276e-01  7.16830199e-01 -6.44056729e+00]
     [-3.62014572e-01  3.46287724e-01  4.48302188e+01]
     [-1.07816433e-04 -6.78525945e-04  1.00000000e+00]]
    X:   #代码复现的透视变换矩阵
    [[ 1.27430276e-01  7.16830199e-01 -6.44056729e+00]
     [-3.62014572e-01  3.46287724e-01  4.48302188e+01]
     [-1.07816433e-04 -6.78525945e-04  1.00000000e+00]]
    '''
```

运行结果如图 5-17 所示。

【例 5-13】 车牌摆正。

解：

(1) 读取彩色图像。



(a) 原图　(b) OpenCV自带函数　(c) 代码复现结果

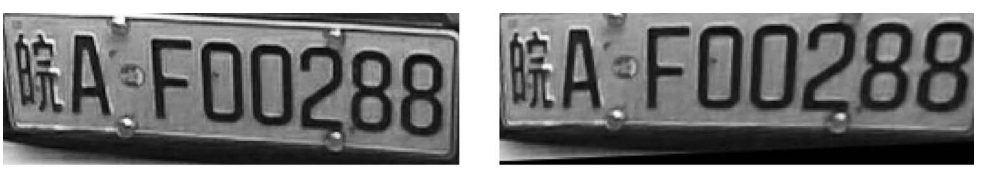
图 5-17　图像透视变换

（2）获取透视变换矩阵。找到原图车牌 4 个角的坐标，设置透视变换后的坐标，调用 OpenCV 实现透视变换矩阵。

（3）显示图像，代码如下：

```
#chapter_13 车牌摆正
import cv2
import numpy as np
import matplotlib.pyplot as plt

#1. 读取彩色图像
img = cv2.imread('pictures/che2.png')
#2. 获取透视变换矩阵
heigh, width, _ = img.shape
#2.1 原图不共线的 4 个点 matSrc,仿射变换后对应的点 matDst
#左上角、左下角、右下角、右上角坐标
src_points = np.float32([[0, 0], [64, 5], [80, 244], [12, 246]])
dst_points = np.float32([[0, 0], [64, 0], [64, 230], [0, 230]])
#2.2.1 调用 OpenCV 实现透视变换矩阵
M = cv2.getPerspectiveTransform(src_points, dst_points)
dst = cv2.warpPerspective(img, M, (width + 10, heigh))
#3. 显示图像
re = np.hstack([img, dst])
plt.imshow(re[..., ::-1])
#cv2.imwrite('pictures/p5_18.jpeg', re)
```

运行结果如图 5-18 所示。



(a) 原图　(b) 透视变换

图 5-18　车牌摆正