東南大學 SOUTHEAST UNIVERSITY 人工智能学院

专业技能实训

授课教师：王洪松

邮箱：hongsongwang@seu.edu.cn

图像变换

几何变换：翻转

- 几何变换是指将一幅图像映射到另外一幅图像内的操作。
- 翻转：dst = cv2.flip(src, flipCode)

dst 代表和原始图像具有同样大小、类型的目标图像。
src 代表要处理的原始图像。
flipCode 代表旋转类型。

flipCode 参数的意义

参数值	说明	意义
0	只能是 0	绕着 x 轴翻转
正数	1、2、3 等任意正数	绕着 y 轴翻转
负数	–1、–2、–3 等任意负数	围绕 x 轴、y 轴同时翻转

几何变换：缩放

- 图像缩放 dst = cv2.resize(src, dsize[, fx[, fy[, interpolation]]])

dst 代表输出的目标图像，该图像的类型与 src 相同，其大小为 dsize（当该值非零时），或者可以通过 src.size()、 fx、 fy 计算得到。
src 代表需要缩放的原始图像。
dsize 代表输出图像大小（width, height），必须为整数。
fx 代表水平方向的缩放比例。fy 代表垂直方向的缩放比例。
interpolation 代表插值方式，默认cv2.INTER_LINEAR。

目标图像的大小可以通过“参数 dsize”或者“参数 fx 和 fy”二者之一来指定。

img.shape[0:2]输出为（height, width）。
OpenCV中， x 表示width或cols, y 表示height或rows。

几何变换：缩放

列数变为原来的 0.9 倍，行数变为原来的 0.5 倍，两种实现

```
rows,cols=img.shape[:2]
size=(int(cols*0.9),int(rows*0.5))
rst=cv2.resize(img, size)
```

```
rst=cv2.resize(img,None,
fx=0.9,fy=0.5)
```

cv2.imread()读取图片之后，数据的形状和维度布局是(H,W,C)，但是使用函数cv2.resize()进行缩放时候，传入的目标形状是(W,H)，而不是(H,W)。

插值方式速度比较：INTER_NEAREST（最近邻插值)> INTER_LINEAR(线性插值)>INTER_CUBIC(三次样条插值)>INTER_AREA（区域插值)
缩小图像时，为了避免出现波纹现象，推荐采用cv2.INTER_AREA 区域插值方法。最近邻插值INTER_NEAREST，一般不推荐使用。
放大图像时，通常使用INTER_CUBIC(速度较慢，但效果最好)，或者使用默认的cv2.INTER_LINEAR(速度较快，效果还可以)。

几何变换：缩放

插值方式列表

类型	说明
cv2.INTER_NEAREST	最临近插值
cv2.INTER_LINEAR	双线性插值（默认方式）
cv2.INTER_CUBIC	三次样条插值。首先对源图像附近的 4×4 近邻区域进行三次样条拟合，然后将目标像素对应的三次样条值作为目标图像对应像素点的值
cv2.INTER_AREA	区域插值，根据当前像素点周边区域的像素实现当前像素点的采样。该方法类似最临近插值方式
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），则将它们设置为零
cv2.WARP_INVERSE_MAP	标志，逆变换。 例如，极坐标变换： • 如果 flag 未被设置，则进行转换：$\mathrm{dst}(\emptyset, \rho) = \mathrm{src}(x, y)$ • 如果 flag 被设置，则进行转换：$\mathrm{dst}(x, y) = \mathrm{src}(\emptyset, \rho)$

几何变换：仿射变换

- 仿射变换是指图像可以通过一系列的几何变换来实现平移、旋转、缩放等多种操作。该变换能够保持图像的平直性和平行性。平直性是指图像经过仿射变换后，直线仍然是直线；平行性是指图像在完成仿射变换后，平行线仍然是平行线。

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

dst 代表仿射后的输出图像。src 代表要仿射的原始图像。
M 代表一个 2×3 的变换矩阵。使用不同的变换矩阵，实现不同的仿射变换。
dsize 决定输出图像的实际大小。flags 代表插值方法，默认INTER_LINEAR。
borderMode 代表边类型，默认为 BORDER_CONSTANT。当该值为 BORDER_TRANSPARENT 时，意味着目标图像内的值不做改变，这些值对应原始图像内的异常值。
borderValue 代表边界值，默认是 0。

几何变换：仿射变换

- 通过转换矩阵 M 将原始图像 src 转换为目标图像 dst

$$\mathrm{dst}(x,y) = \mathrm{src}(M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23})$$

- 平移变换

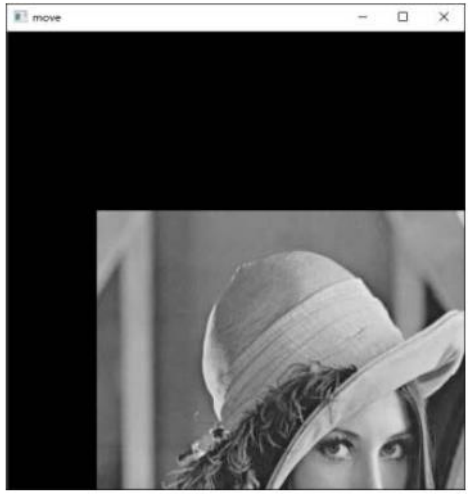
$$\mathrm{dst}(x,y) = \mathrm{src}(1 \cdot x + 0 \cdot y + M_{13}, 0 \cdot x + 1 \cdot y + M_{23})$$

对应的变换矩阵为

$$M = \begin{bmatrix} 1 & 0 & M_{13} \\ 0 & 1 & M_{23} \end{bmatrix}$$

```
height,width=img.shape[:2]
x, y =100, 200
M = np.float32([[1, 0, x], [0, 1, y]])
move=cv2.warpAffine(img,M,(width,height))
```

原始图像



移动结果图像

几何变换：仿射变换

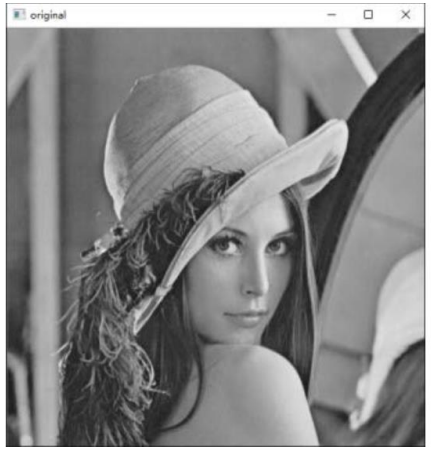
- 旋转变换

$$\mathrm{dst}(x,y) = \mathrm{src}(M_{11}x + M_{12}y, M_{21}x + M_{22}y)$$

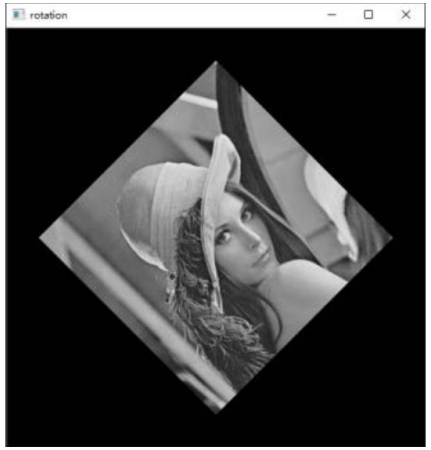
可以通过cv2.getRotationMatrix2D()函数获取变换矩阵
retval=cv2.getRotationMatrix2D(center, angle, scale)

center 为旋转的中心点。scale 为变换尺度（缩放大小）。
angle 为旋转角度，正数表示逆时针旋转，负数表示顺时针旋转。

```
height,width=img.shape[:2]
M=cv2.getRotationMatrix2D((
width/2,height/2),45,0.6)
rotate=cv2.warpAffine(img,M,
(width,height))
```



原始图像



选择结果图像

几何变换：仿射变换

- 复杂放射变换

可以通过cv2.getAffineTransform()函数获取变换矩阵
retval=cv2.getAffineTransform(src, dst)

src 代表输入图像的三个点坐标。dst 代表输出图像的三个点坐标。
上述参数通过函数cv2.getAffineTransform()定义了两个平行四边形。
src 和 dst 中的三个点分别对应平行四边形的左上角、右上角、左下角三个点。

```
p1=np.float32([[0,0],[cols-1,0],[0,rows-1]])
p2=np.float32([[0,rows*0.33],[cols*0.85,rows*0.25],[cols*0.15,rows*0.7]])
M=cv2.getAffineTransform(p1,p2)
dst=cv2.warpAffine(img,M,(cols,rows))
```

几何变换：透视变换

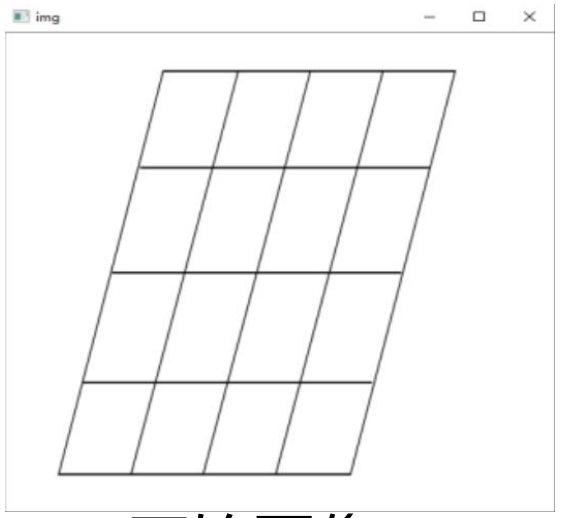
- 仿射变换可以将矩形映射为任意平行四边形，透视变换则可以将矩形映射为任意四边形。

透视变换函数
dst = cv2.warpPerspective(src, M, dsize[, flags[, borderMode[, borderValue]]])
和仿射变换不同，M 代表一个 3×3 的变换矩阵。

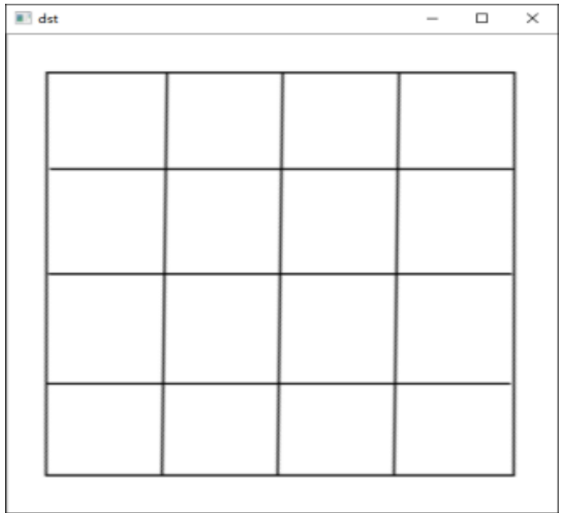
使用函数 cv2.warpPerspective()生成所使用的转换矩阵
retval = cv2.getPerspectiveTransform(src, dst)
src 代表输入图像的四个顶点的坐标。
dst 代表输出图像的四个顶点的坐标。

几何变换：透视变换

```
import cv2
import numpy as np
img=cv2.imread('demo.bmp')
rows,cols=img.shape[:2]
print(rows,cols)
pts1 = np.float32([[150,50],[400,50],[60,450],[310,450]])
pts2 = np.float32([[50,50],[rows-50,50],[50,cols-
50],[rows-50,cols-50]])
M=cv2.getPerspectiveTransform(pts1,pts2)
dst=cv2.warpPerspective(img,M,(cols,rows))
cv2.imshow("img",img)
cv2.imshow("dst",dst)
cv2.waitKey()
cv2.destroyAllWindows()
```



原始图像



透视结果图像

重映射

- 重映射：把一幅图像内的像素点放置到另外一幅图像内的指定位置。
- OpenCV使用自定义的方式来完成重映射函数 cv2.remap。

dst = cv2.remap(src, map1, map2, interpolation[, borderMode[, borderValue]])

src 代表原始图像。dst 代表目标图像。
map1 参数有两种可能的值：

- 表示(x,y)点的一个映射。
- 表示 CV_16SC2 , CV_32FC1, CV_32FC2 类型(x,y)点的 x 值。

map2 参数同样有两种可能的值：

- 当 map1 表示(x,y)时，该值为空。
- 当 map1 表示(x,y)点的 x 值时，该值是 CV_16UC1, CV_32FC1 类型(x,y)点的 y 值。

Interpolation 代表插值方式，见cv2.resize。borderMode 代表边界模式。

图像非线性变换

非线性变换：图像特效

- 挤压（凹透镜滤镜效果）
  - 将图像向内挤压，挤压的过程产生压缩变形。
  - 挤压效果的实现是通过极坐标的形式，设图像中心为O(x,y)，某点距离中心O的距离为半径R，非线性方式改变半径R但不改变点的方向，就构成了图像挤压。也可以自定义加压中心点，计算半径方式相同。
- 扩张（凸透镜滤镜效果）
  - 实现k的根号与k的比值，sqrt(k)/k， 当k为1时总倍率为1，当k小于1时，总倍率为渐变倍率。
- 扭曲
  - 对图像的像素坐标进行正弦变换，映射到对应坐标就完成了图像扭曲。
- 漩涡
  - 涡流，把图像变换成流动的漩涡。

凹透镜滤镜效果

```
def filter_concave_lens(src):
    width = src.shape[1]
    heigh = src.shape[0]
    center = (width // 2, heigh // 2)
    img2 = np.zeros(src.shape, dtype=np.uint8)
    for y in range(heigh):
        for x in range(width):
            theta = np.arctan2(y - center[1], x - center[0])
            R2 = int(np.sqrt(np.linalg.norm(np.array([x, y]) - np.array(center))) * 8)
            newX = center[0] + int(R2 * np.cos(theta))
            newY = center[1] + int(R2 * np.sin(theta))
            if newX < 0:
                newX = 0
            elif newX >= width:
                newX = width - 1
            if newY < 0:
                newY = 0
            elif newY >= heigh:
                newY = heigh - 1
            img2[y, x] = src[newY, newX]
    return img2
```

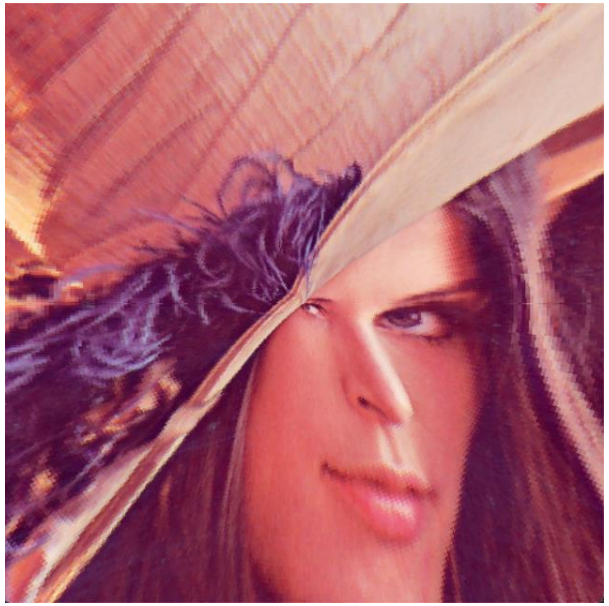
（1）确定一个中心点。

（2）根据图像位置和中心点的位置关系，确定一个转换角度。

（3）利用公式，获得新的图像像素坐标。

（4）将原图中的像素值映射到新的图像中。

原始图像

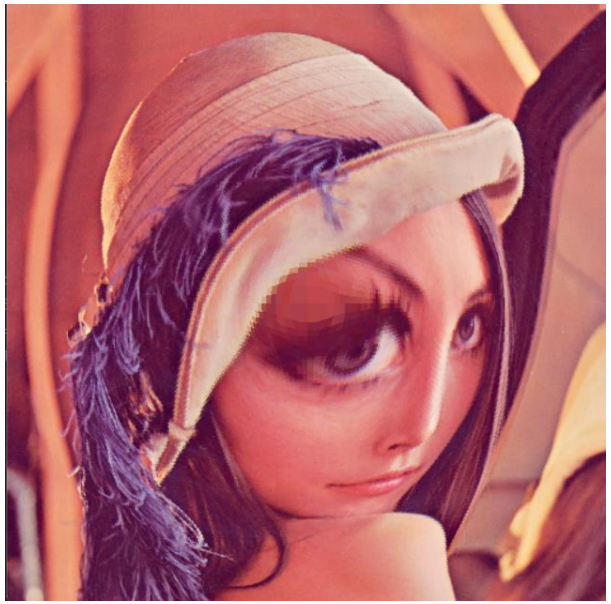


凹透镜效果图像

凸透镜滤镜效果

```
def filter_convex lens(src_img):
    row=src_img.shape[0]
    col=src_img.shape[1]
    channel=src_img.shape[2]
    new_img=np.zeros([row,col,channel],dtype=np.uint8)
    center_x=row/2
    center_y=col/2
    # radius=math.sqrt(center_x*center_x+center_y*center_y)/2
    radius = min(center_x,center_y)
    for i in range(row):
        for j in range(col):
            distance=((i-center_x)*(i-center_x)+(j-center_y)*(j-center_y))
            new_dist=math.sqrt(distance)
            new_img[i,j,:]=src_img[i,j,:]
            if distance<=radius**2:
                new_i=np.int(np.floor(new_dist*(i-center_x)/radius+center_x))
                new_j=np.int(np.floor(new_dist*(j-center_y)/radius+center_y))
                new_img[i,j,:]=src_img[new_i,new_j,:]
    return new_img
```

原始图像

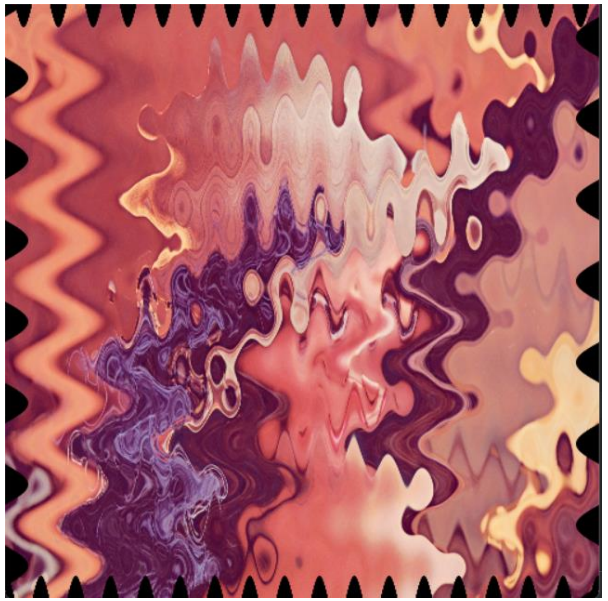


凸透镜效果图像

正弦变换效果

```
row, col, channel = img.shape
alpha = 70.0
beta = 30.0
degree = 20.0
center_x = (col-1)/2.0
center_y = (row-1)/2.0
y_mask, x_mask = np.indices((row,col))
xx_dif = x_mask - center_x
yy_dif = center_y - y_mask
x = degree * np.sin(2 * math.pi * yy_dif / alpha) + xx_dif
y = degree * np.cos(2 * math.pi * xx_dif / beta) + yy_dif
x_new = x + center_x
y_new = center_y - y
x_new = x_new.astype(np.float32)
y_new = y_new.astype(np.float32)
dst = cv2.remap(img, x_new, y_new, cv2.INTER_LINEAR)
```

原始图像



正弦变换效果图像

正弦变换效果

x方向正弦变换效果图像

y方向正弦变换效果图像

漩涡滤镜效果

```
row, col, channel = img.shape
img_out = img * 1.0
degree = 70
center_x = (col - 1) / 2.0
center_y = (row - 1) / 2.0
y_mask, x_mask = np.indices((row, col))
xx_dif = x_mask - center_x
yy_dif = center_y - y_mask
r = np.sqrt(xx_dif * xx_dif + yy_dif * yy_dif)
theta = np.arctan(yy_dif / xx_dif)
mask_1 = xx_dif < 0
theta = theta * (1 - mask_1) + (theta + math.pi) * mask_1
theta = theta + r / degree
x_new = r * np.cos(theta) + center_x
y_new = center_y - r * np.sin(theta)
x_new = x_new.astype(np.float32)
y_new = y_new.astype(np.float32)
dst = cv2.remap(img, x_new, y_new, cv2.INTER_LINEAR )
```

原始图像

旋涡滤镜效果图像

打马赛克

- 将特定区域的细节模糊，造成色块打乱的效果。
- 把图像上某个像素点一定范围邻域内的所有点用邻域内随机选取的一个像素点的颜色代替。模糊细节，保留大体的轮廓。

```
img_out = img.copy()
row, col, channel = img.shape
half_patch =10
for i in range(half_patch, row-1-half_patch, half_patch):
    for j in range (half_patch, col-1-half_patch, half_patch):
        k1 = random.random() - 0.5
        k2 = random.random() - 0.5
        m=np.floor(k1*(half_patch*2 + 1))
        n=np.floor(k2*(half_patch*2 + 1))
        h=int((i+m) % row)
        w=int((j+n) % col)

        img_out[i-half_patch:i+half_patch, j-half_patch:j+half_patch, :] =\
                    img[h, w, :]
```

专业技能实训：练习题

- 图像仿射变换
  - 输入图像，顺时针依次用鼠标点击三个点，作为仿射变换后和原图左上、右上、右下三个点对应的点。
  - 按下Enter键，对图像做放射变换。
  - 当所选择的三个点组成的不是长方形时，将边界黑色背景区域设置为透明，保存新的图像。
- 图像透视变换
  - 输入图像，顺时针依次用鼠标点击四个点，作为仿射变换后和原图左上、右上、右下、左下四个点对应的点。
  - 按下Enter键，对图像做透视变换。
  - 将边界黑色背景区域设置为透明，保存新的图像。

专业技能实训：练习题

- 凸透镜滤镜效果
  - 输入图像，采用两个滚动条控制变换的中心的坐标。
  - 采用滚动条控制变换倍率。
  - 采用凸透镜滤镜效果转化图像（用numpy向量运算实现，不用for循环）。
  - 对比显示原图和处理后的图像。

- 凹透镜滤镜效果
  - 输入图像，采用两个滚动条控制变换的中心的坐标。
  - 采用滚动条控制变换倍率。
  - 采用凹透镜滤镜效果转化图像（用numpy向量运算实现，不用for循环）。
  - 对比显示原图和处理后的图像。

专业技能实训：练习题

- 正弦变换效果
  - 采用4个滚动条分别控制x和y方向的变换倍率和正弦波周期。
  - 拖动滚动条实现对原图的正弦变换。
  - 对比显示原图和处理后的图像。

- 漩涡滤镜效果
  - 输入图像，采用两个滚动条控制变换的中心的坐标。
  - 采用滚动条变换倍率。
  - 拖动滚动条实现对原图的漩涡滤镜变换。
  - 对比显示原图和处理后的图像。

专业技能实训：练习题

- 对图像区域打马赛克
  - 输入图像，用cv2.selectROI选择矩形区域。
  - 对所选择的图像区域打马赛克。

- 把图像变换到扇环中
  - 用三个滚动条分别控制扇环的内外半径和扇形的角度（e.g., 0~360度），扇环如右图所示。
  - 将图像映射到扇环中。