

# 图像的几何变换

电子信息工程系

袁羽

# 目录


CONTENTS

- 1 缩放
- 2 翻转
- 3 仿射变换
- 4 透视



# 图像的几何变换

图像几何变换可以改变图像的几何结构，例如大小、角度和形状等，让图像呈现出缩放（灰度插值）、翻转、映射和透视效果（坐标变换，本质上是将一幅图像中的坐标位置映射到另一幅图像中新的坐标位置）。这些几何变换操作都涉及复杂、精密的计算，OpenCV将这些计算过程封装成非常灵活的方法，只需修改一些参数，就能实现图像的变换效果。



# 一 缩放

“缩”表示缩小，“放”表示放大，通过OpenCV提供的resize()方法可以随意更改图像的大小比例，其语法如下：`dst = cv2.resize(src, dsize, fx, fy, interpolation)`

参数说明：

src：原始图像。

dsize：输出图像的大小，格式为（宽，高），单位为像素。

fx：可选参数。水平方向的缩放比例。

fy：可选参数。垂直方向的缩放比例。

interpolation：可选参数。缩放的插值方式。在图像缩小或放大时需要删减或补充像素，该参数可以指定使用哪种算法对像素进行增减。建议使用默认值。

返回值说明：

dst：缩值之后的图像。

# 一 缩放

interpolation: 可选参数。缩放的插值方式。在图像缩小或放大时需要删减或补充像素, 该参数可以指定使用哪种算法对像素进行增减。建议使用默认值。

<code>cv2.INTER_NEAREST</code>	最近邻插值
<code>cv2.INTER_LINEAR</code>	双线性插值
<code>cv2.INTER_CUBIC</code>	三次样条插值
<code>cv2.INTER_AREA</code>	使用像素区域关系重新采样。它可能是图像抽取的首选方法, 因为它可以提供无莫尔条纹的结果。但是当图像被缩放时, 它类似于 <code>INTER_NEAREST</code> 方法。

通常的, 缩小使用 `cv2.INTER_AREA`, 放大使用 `cv2.INTER_CUBIC`(较慢), `cv2.INTER_LINEAR`(较快效果也不错)。默认情况下, 所有的放缩都使用 `cv2.INTER_LINEAR`。

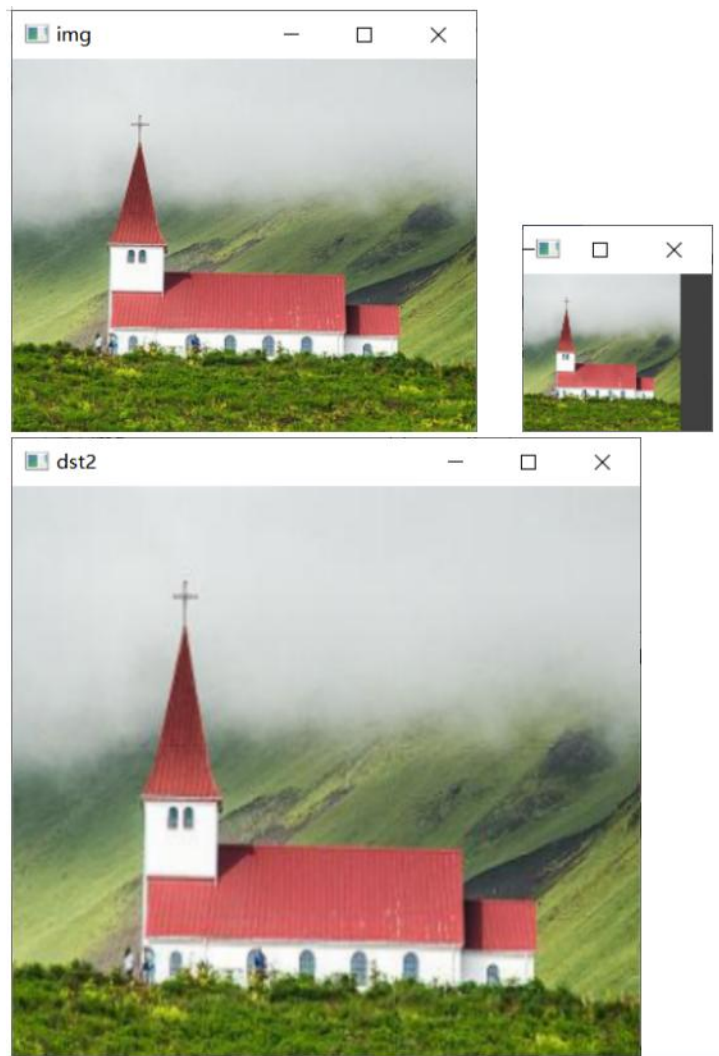
# 一 缩放

`resize()`方法有两种使用方式，一种是通过`dsize`参数实现缩放，另一种是通过`fx`和`fy`参数实现缩放。

## 1. `dsize`参数实现缩放

`dsize`参数的格式是一个元组，例如`(100, 200)`，表示将图像按照宽100像素、高200像素的大小进行缩放。如果使用`dsize`参数，就可以不写`fx`和`fy`参数。

例 将图像按照指定宽高进行缩放：将一个图像按照宽100像素、高100像素的大小进行缩小，再按照宽400像素、高400像素的大小进行放大。



**例 将图像按照指定宽高进行缩放：将一个图像按照宽100像素、高100像素的大小进行缩小，再按照宽400像素、高400像素的大小进行放大。**

```
import cv2

img = cv2.imread("t.png") # 读取图像

dst1 = cv2.resize(img, (100, 100)) # 按照宽100像素、高100像素的大小进行缩放
dst2 = cv2.resize(img, (400, 400)) # 按照宽400像素、高400像素的大小进行缩放

cv2.imshow("img", img) # 显示原图

cv2.imshow("dst1", dst1) # 显示缩放图像

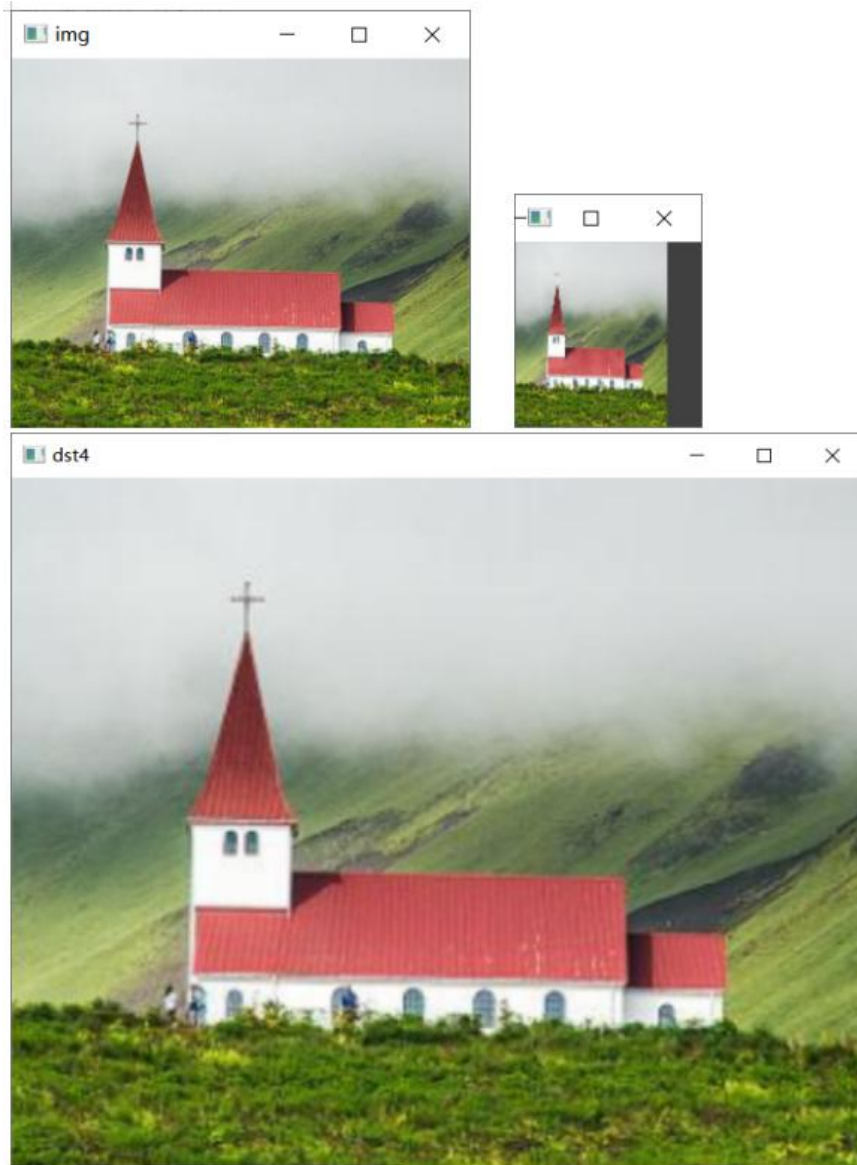
cv2.imshow("dst2", dst2)

cv2.waitKey() # 按下任何键盘按键后

cv2.destroyAllWindows() # 释放所有窗体
```



例 将图像按照指定比例进行缩放：将一个图像宽缩小到原来的 $1/3$ 、高缩小到原来的 $1/2$ ，再将图像宽放大2倍，高也放大2倍。



**例 将图像按照指定比例进行缩放：将一个图像宽缩小到原来的1/2、高缩小到原来的1/2，再将图像宽放大2倍，高也放大2倍。**

```
import cv2

img = cv2.imread("t.png") # 读取图像

dst3 = cv2.resize(img, None, fx=1 / 2, fy=1 / 2) # 将宽缩小到原来的1/2、高缩小到原来的1/2

dst4 = cv2.resize(img, None, fx=2, fy=2) # 将宽高扩大2倍

cv2.imshow("img", img) # 显示原图

cv2.imshow("dst3", dst3) # 显示缩放图像

cv2.imshow("dst4", dst4) # 显示缩放图像

cv2.waitKey() # 按下任何键盘按键后

cv2.destroyAllWindows() # 释放所有窗体
```

练 在图像中截取一块子图，使用不同的插值方法放大子图，比较不同插值方法的效果。

```
import cv2

lenna = cv2.imread("lenna.jpg")
eye = lenna[230:280,310:360,:]

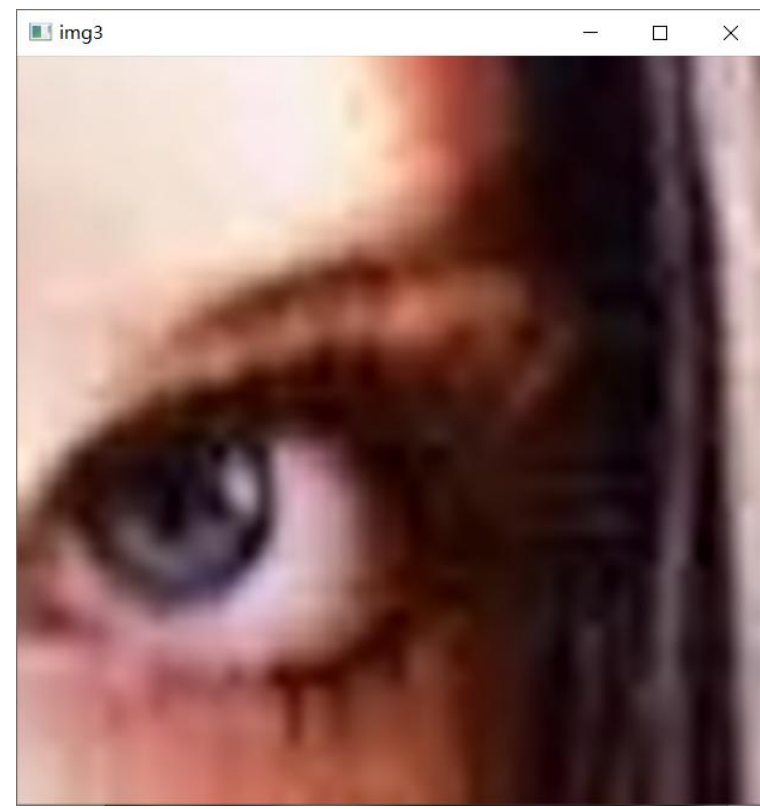
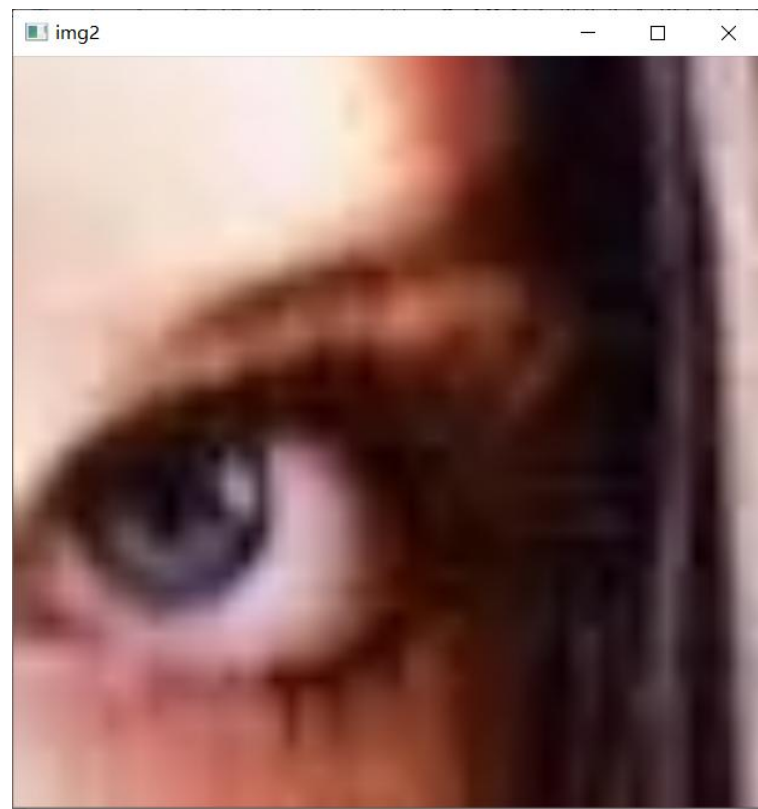
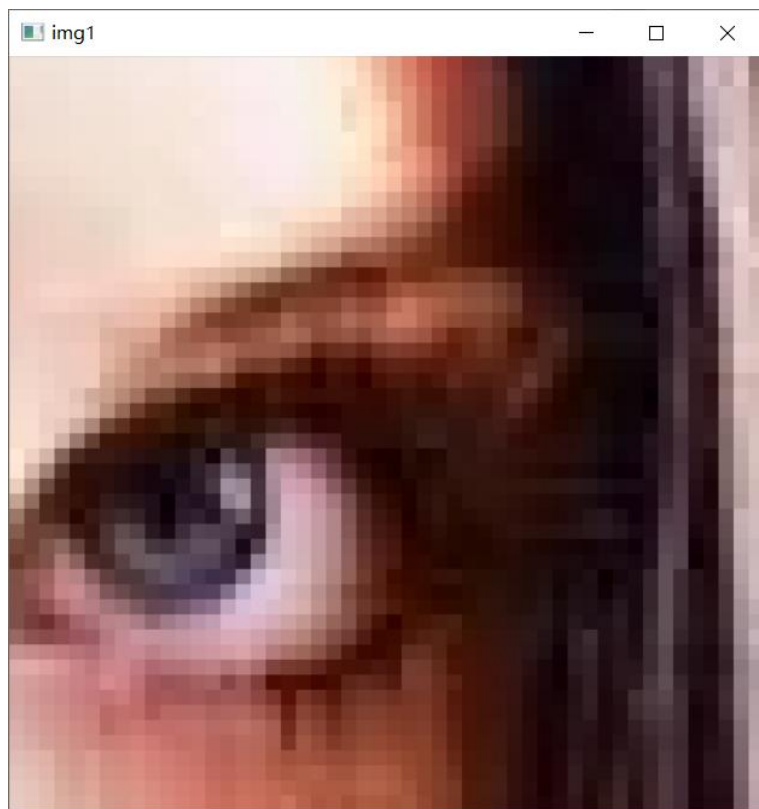
img1 = cv2.resize(eye,None,fx=10,fy=10,interpolation=cv2.INTER_NEAREST)
img2 = cv2.resize(eye,None,fx=10,fy=10,interpolation=cv2.INTER_LINEAR)
img3 = cv2.resize(eye,None,fx=10,fy=10,interpolation=cv2.INTER_CUBIC)

cv2.imshow("lenna", lenna)
cv2.imshow("eye", eye)
cv2.imshow("img1", img1)
cv2.imshow("img2", img2)
cv2.imshow("img3", img3)
cv2.waitKey()
cv2.destroyAllWindows()
```

练 在图像中截取一块子图，使用不同的插值方法放大子图，比较不同插值方法的效果。

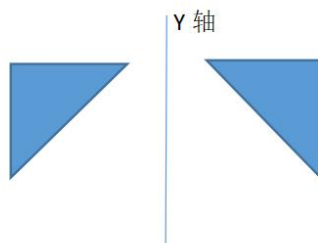
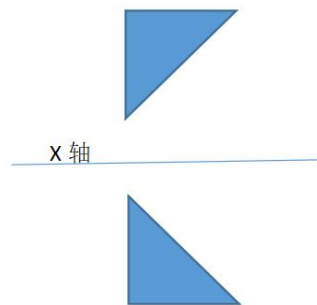


练 在图像中截取一块子图，使用不同的插值方法放大子图，比较不同插值方法的效果。



## 二 翻转

水平方向被称为X轴，垂直方向被称为Y轴。图像沿着X轴或Y轴翻转之后，可以呈现出镜面或倒影的效果



## 二 翻转

OpenCV通过cv2.flip()方法实现翻转效果，其语法如下：

```
dst = cv2.flip(src, flipCode)
```

参数说明：

src：原始图像。

flipCode：翻转类型，

flipCode为0表示沿着X轴旋转

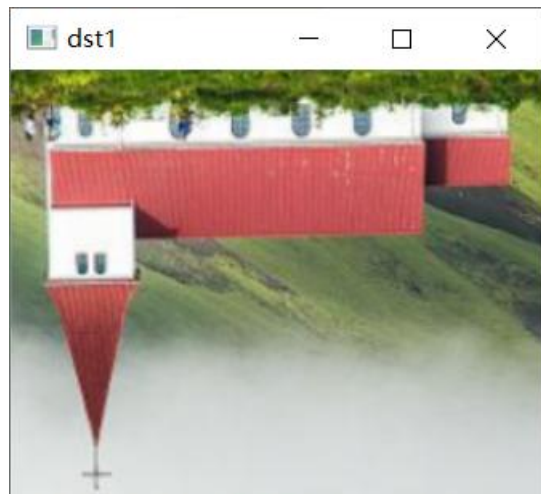
flipCode为正数表示沿着Y轴旋转

flipCode为负数表示沿着X，Y轴旋转

返回值说明：

dst：翻转之后的图像。

例 同时实现3种翻转效果：分别让图像沿X轴翻转，沿Y轴翻转，同时沿X轴、Y轴翻转。





**例 同时实现3种翻转效果：分别让图像沿X轴翻转，沿Y轴翻转，同时沿X轴、Y轴翻转。**

```
import cv2

img = cv2.imread("t.png") # 读取图像

dst1 = cv2.flip(img, 0) # 沿X轴翻转

dst2 = cv2.flip(img, 1) # 沿Y轴翻转

dst3 = cv2.flip(img, -1) # 同时沿X轴、Y轴翻转

cv2.imshow("img", img) # 显示原图

cv2.imshow("dst1", dst1) # 显示翻转之后的图像

cv2.imshow("dst2", dst2)

cv2.imshow("dst3", dst3)

cv2.waitKey() # 按下任何键盘按键后

cv2.destroyAllWindows() # 释放所有窗体
```

# 例 镜像拼接效果



# 例 镜像拼接效果

```
import cv2

import numpy as np

img = cv2.imread("t.png") # 读取图像

img1 = cv2.flip(img, 0) # 沿X轴翻转

img2 = cv2.flip(img, 1) # 沿Y轴翻转

dst1 = np.vstack((img, img1)) #垂直拼接

dst2 = np.hstack((img, img2)) #水平拼接

cv2.imshow("dst1", dst1) # 显示翻转之后的图像

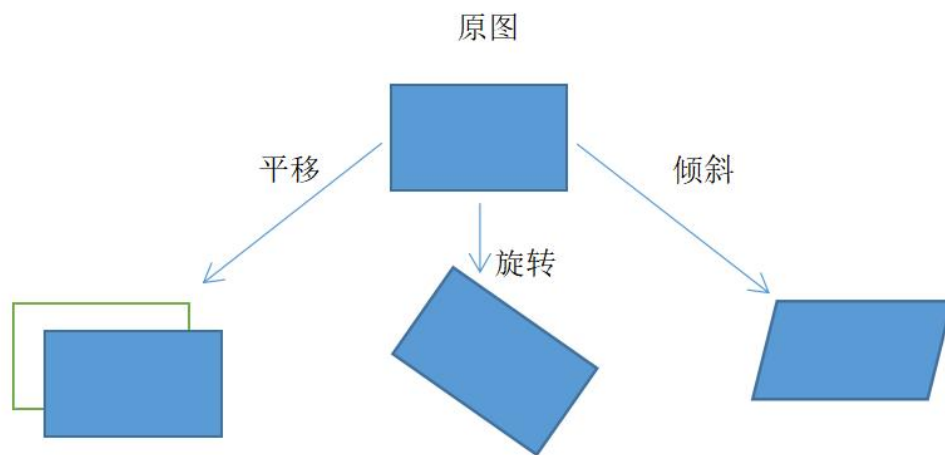
cv2.imshow("dst2", dst2)

cv2.waitKey() # 按下任何键盘按键后

cv2.destroyAllWindows() # 释放所有窗体
```

# 三 仿射变换

仿射变换是一种仅在二维平面中发生的几何变形，变换之后的图像仍然可以保持直线的“平直性”和“平行性”，也就是说原来的直线变换之后还是直线，平行线变换之后还是平行线。常见的仿射变换效果包含平移、旋转和倾斜。



# 三 仿射变换

OpenCV通过cv2.warpAffine()方法实现仿射变换效果，其语法如下：

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

参数说明：

src：原始图像。

M：一个2行3列的矩阵，根据此矩阵的值变换原图中的像素位置。

dsize：输出图像的尺寸大小。

flags：可选参数，插值方式，建议使用默认值。

borderMode：可选参数，边界类型，建议使用默认值。

borderValue：可选参数，边界值，默认为0，建议使用默认值。

返回值说明：

dst：经过仿射变换后输出图像。

# 三 仿射变换

M也被叫作仿射矩阵，实际上就是一个 $2 \times 3$ 的列表，其格式如下：

$$M = [[a, b, c],[d, e, f]]$$

图像做何种仿射变换，完全取决于M的值，仿射变换输出的图像按照以下公式进行计算：

$$\text{新}x = \text{原}x \times a + \text{原}y \times b + c$$

$$\text{新}y = \text{原}x \times d + \text{原}y \times e + f$$

原x和原y表示原始图像中像素的横坐标和纵坐标，新x与新y表示同一个像素经过仿射变换后在新图像中的横坐标和纵坐标。

# 三 仿射变换

M矩阵中的数字采用32位浮点格式，可以采用两种方式创建M。

(1) 创建一个全是0的M，代码如下：

```
import numpy as np
```

```
M = np.zeros((2, 3), np.float32)
```

(2) 创建M的同时赋予具体值，代码如下：

```
import numpy as np
```

```
M = np.float32([[1, 2, 3], [4, 5, 6]])
```

通过设定M的值就可以实现多种仿射效果，下面分别介绍如何实现图像的平移、旋转和倾斜。

# 三 仿射变换

1. 平移就是让图像中的所有像素同时沿着水平或垂直方向移动。实现这种效果只需要将M的值按照以下格式进行设置：

$$M = [[1, 0, \text{水平移动的距离}], [0, 1, \text{垂直移动的距离}]]$$

原始图像的像素就会按照以下公式进行变换：

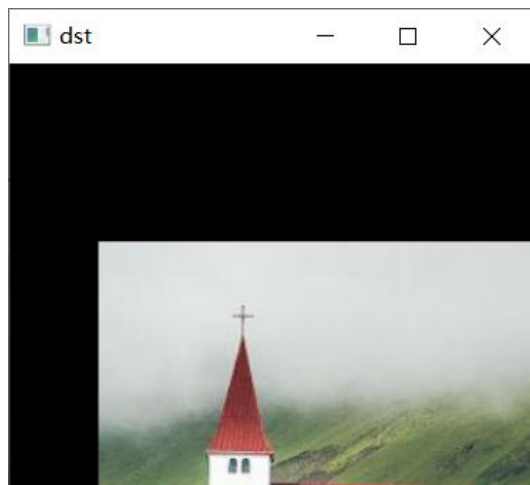
$$\text{新}x = \text{原}x \times 1 + \text{原}y \times 0 + \text{水平移动的距离} = \text{原}x + \text{水平移动的距离}$$

$$\text{新}y = \text{原}x \times 0 + \text{原}y \times 1 + \text{垂直移动的距离} = \text{原}y + \text{垂直移动的距离}$$

若水平移动的距离为正数，图像向右移动，若为负数，图像向左移动；若垂直移动的距离为正数，图像向下移动，若为负数，图像向上移动；若水平移动的距离和垂直移动的距离的值为0，图像不发生移动。



例 让图像向右下方平移：向右移动50像素、向下移动100像素



## 例 让图像向右下方平移：向右移动50像素、向下移动100像素

```
import cv2

import numpy as np

img = cv2.imread("t.png") # 读取图像

rows,cols =img.shape[:2]

M = np.float32([[1, 0, 50], # 横坐标向右移动50像素
                [0, 1, 100]]) # 纵坐标向下移动100像素

dst = cv2.warpAffine(img, M, (cols, rows))

cv2.imshow("img", img) # 显示原图

cv2.imshow("dst", dst) # 显示仿射变换效果

cv2.waitKey() # 按下任何键盘按键后

cv2.destroyAllWindows() # 释放所有窗体
```

## 练 实现其他平移效果。

通过修改M的值可以实现其他平移效果。例如：

横坐标不变，纵坐标向上移动50像素，M的值如下：

```
M = np.float32([[1, 0, 0], # 横坐标不变  
               [0, 1, -50]]) # 纵坐标向上移动50像素
```

纵坐标不变，横坐标向左移动200像素，M的值如下：

```
M = np.float32([[1, 0, -200], # 横坐标向左移动200像素  
               [0, 1, 0]]) # 纵坐标不变
```

# 三 仿射变换

## 2. 旋转

让图像旋转也是通过M矩阵实现的，但得出这个矩阵需要做很复杂的运算，于是OpenCV提供了getRotationMatrix2D()方法自动计算旋转图像的M矩阵。

getRotationMatrix2D()方法的语法如下：

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

参数说明：

center：旋转的中心点坐标。

angle：旋转的角度（不是弧度）。正数表示逆时针旋转，负数表示顺时针旋转。

scale：缩放比例，浮点类型。如果取值1，表示图像保持原来的比例。

返回值说明：

M：getRotationMatrix2D()方法计算出的仿射矩阵。

例 让图像逆时针旋转 $30^\circ$  的同时缩小到原来的80%



## 例 让图像逆时针旋转 $30^\circ$ 的同时缩小到原来的80%

```
import cv2

img = cv2.imread("t.png") # 读取图像

rows,cols =img.shape[:2]

center = (rows / 2, cols / 2) # 图像的中心点

M = cv2.getRotationMatrix2D(center, 30, 0.8) # 以图像为中心，逆时针旋转30度，
缩放0.8倍

dst = cv2.warpAffine(img, M, (cols, rows)) # 按照M进行仿射

cv2.imshow("img", img) # 显示原图

cv2.imshow("dst", dst) # 显示仿射变换效果

cv2.waitKey() # 按下任何键盘按键后

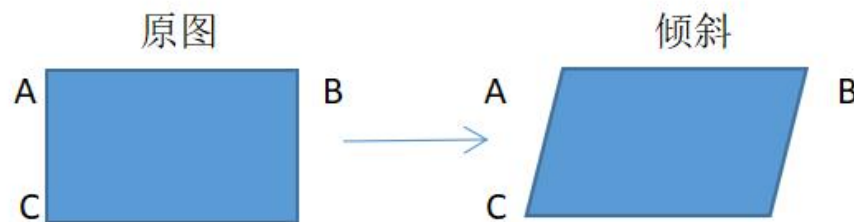
cv2.destroyAllWindows() # 释放所有窗体
```

# 三 仿射变换

## 3. 倾斜

OpenCV需要定位图像的3个点来计算倾斜效果，这3个点分别是“左上角”点A、“右上角”点B和“左下角”点C。OpenCV会根据这3个点的位置变化来计算其他像素的位置变化。因为要保证图像的“平直性”和“平行性”，所以不需要“右下角”的点做第4个参数，右下角这个点的位置根据A、B、C 3点的变化自动计算得出。

“平直性”是指图像中的直线在经过仿射变换之后仍然是直线。“平行性”是指图像中的平行线在经过仿射变换之后仍然是平行线。



# 三 仿射变换

## 3. 倾斜

让图像倾斜也是需要通过M矩阵实现的，但得出这个矩阵需要做很复杂的运算，于是OpenCV提供了getAffineTransform()方法来自动计算倾斜图像的M矩阵。

getRotationMatrix2D()方法的语法如下：

```
M = cv2.getAffineTransform(src, dst)
```

参数说明：

src：原图3个点坐标，格式为3行2列的32位浮点数列表，例如：[[0, 1], [1, 0], [1, 1]]。

dst：倾斜图像的3个点坐标，格式与src一样。

返回值说明：

M：getAffineTransform()方法计算出的仿射矩阵。



例 让图像向右倾斜。



## 例 让图像向右倾斜。

```
import cv2
import numpy as np

img = cv2.imread("t.png") # 读取图像
rows, cols = img.shape[:2] # 图像像素行列数
p1 = np.zeros((3, 2), np.float32) # 32位浮点型空列表, 原图三个点
p1[0] = [0, 0] # 左上角点坐标
p1[1] = [cols - 1, 0] # 右上角点坐标
p1[2] = [0, rows - 1] # 左下角点坐标
p2 = np.zeros((3, 2), np.float32) # 32位浮点型空列表, 倾斜图三个点
p2[0] = [50, 0] # 左上角点坐标, 向右挪50像素
p2[1] = [cols - 1, 0] # 右上角点坐标, 位置不变
p2[2] = [0, rows - 1] # 左下角点坐标, 位置不变
```

```
M = cv2.getAffineTransform(p1, p2) # 根据三个点的变化轨迹计算出M矩阵
dst = cv2.warpAffine(img, M, (cols, rows)) # 按照M进行仿射
cv2.imshow('img', img) # 显示原图
cv2.imshow('dst', dst) # 显示仿射变换效果
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 释放所有窗体
```

## 例 让图像向右倾斜。

```
p1 = np.zeros((3, 2), np.float32) # 32位浮点型空列表, 原图三个点
```

```
p1[0] = [0, 0] # 左上角点坐标
```

```
p1[1] = [cols - 1, 0] # 右上角点坐标
```

```
p1[2] = [0, rows - 1] # 左下角点坐标
```

以上可简写为:

```
p1 = np.float32([[0, 0],[cols - 1, 0],[0, rows - 1]])
```

```
p2 = np.zeros((3, 2), np.float32) # 32位浮点型空列表, 倾斜图三个点
```

```
p2[0] = [50, 0] # 左上角点坐标, 向右挪50像素
```

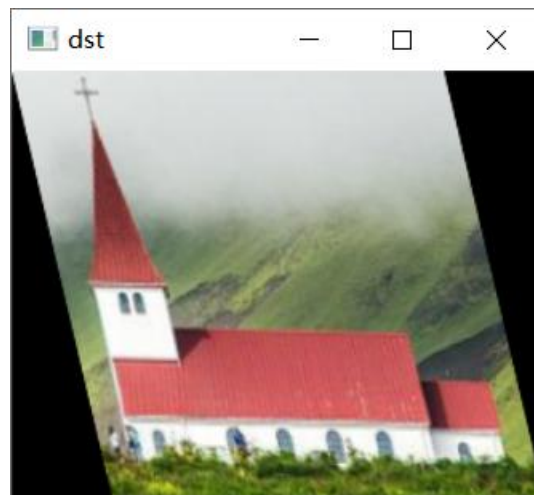
```
p2[1] = [cols - 1, 0] # 右上角点坐标, 位置不变
```

```
p2[2] = [0, rows - 1] # 左下角点坐标, 位置不变
```

以上可简写为:

```
p2 = np.float32([[50, 0],[cols - 1, 0],[0, rows - 1]])
```

例 让图像向左倾斜。



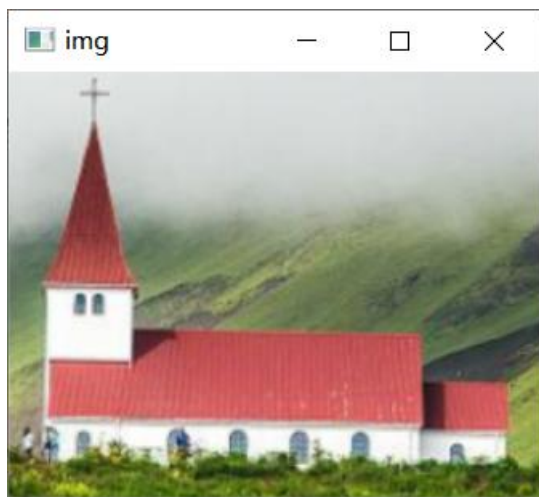
## 例 让图像向左倾斜。

如果让图像向左倾斜，不能只通过移动点A来实现，还需要通过移动点B和点C来实现，3个点的修改方式如下：

```
p1 = np.zeros((3, 2), np.float32) # 32位浮点型空列表，原图三个点
p1[0] = [0, 0] # 左上角点坐标
p1[1] = [cols - 1, 0] # 右上角点坐标
p1[2] = [0, rows - 1] # 左下角点坐标
p2 = np.zeros((3, 2), np.float32) # 32位浮点型空列表，倾斜图三个点
p2[0] = [0, 0] # 左上角点坐标，向右挪50像素
p2[1] = [cols - 1 - 50, 0] # 右上角点坐标，位置不变
p2[2] = [50, rows - 1] # 左下角点坐标，位置不变
使用这两组数据计算出的M矩阵可以实现向左倾斜效果。
```

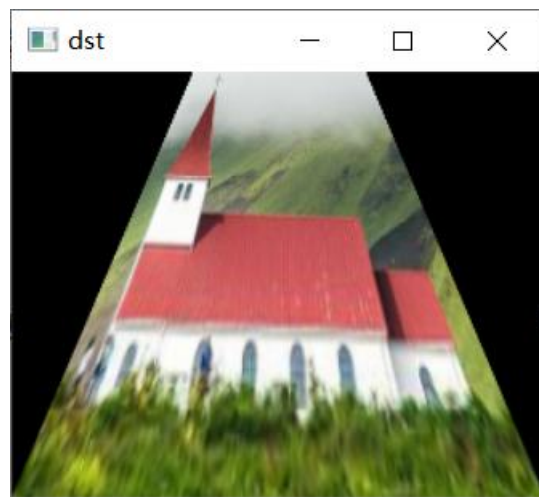
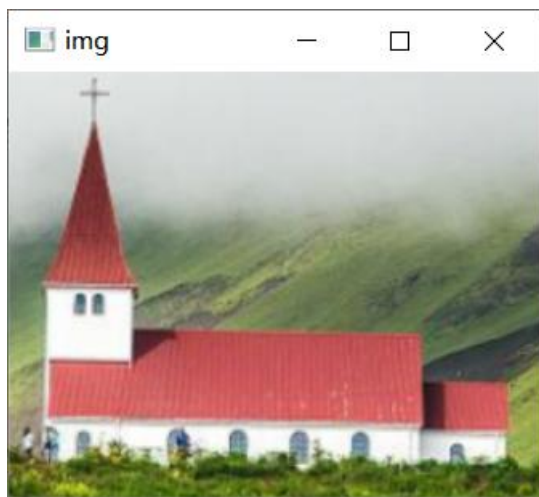
# 四 透视

仿射是让图像在二维平面中变形，透视是让图像在三维空间中变形。从不同的角度观察物体，会看到不同的变形画面，例如，矩形会变成不规则的四边形，直角会变成锐角或钝角，圆形会变成椭圆，等等，这种变形之后的画面就是透视图。



# 四 透视

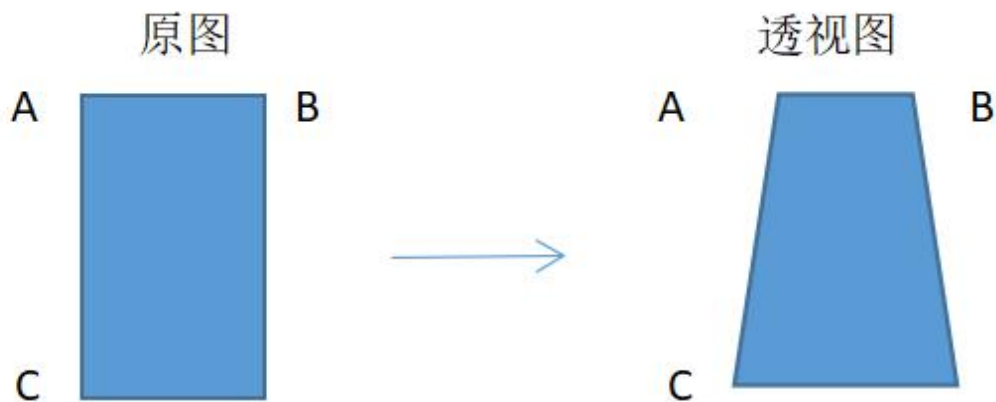
仿射是让图像在二维平面中变形，透视是让图像在三维空间中变形。从不同的角度观察物体，会看到不同的变形画面，例如，矩形会变成不规则的四边形，直角会变成锐角或钝角，圆形会变成椭圆，等等，这种变形之后的画面就是透视图。



从图像的底部观察图，眼睛距离图像底部较近，所以图像底部宽度不变，但眼睛距离图像顶部较远，图像顶部宽度就会等比缩小，于是观察者就会看到如图所示的透视效果。

# 四 透视

OpenCV中需要通过定位图像的4个点计算透视效果，OpenCV根据这4个点的位置变化来计算其他像素的位置变化。透视效果不能保证图像的“平直性”和“平行性”。





## 四 透视

OpenCV通过warpPerspective()方法来实现透视效果，其语法如下：

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

参数说明：

src：原始图像。

M：一个3行3列的矩阵，根据此矩阵的值变换原图中的像素位置。

dsize：输出图像的尺寸大小。

flags：可选参数，插值方式，建议使用默认值。

borderMode：可选参数，边界类型，建议使用默认值。

borderValue：可选参数，边界值，默认为0，建议使用默认值。

返回值说明：

dst：经过透视变换后输出图像。

## 四 透视

warpPerspective()方法也需要通过M矩阵计算透视效果，但得出这个矩阵需要做很复杂的运算，于是OpenCV提供了getPerspectiveTransform()方法自动计算M矩阵。

getPerspectiveTransform()方法的语法如下：

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

参数说明：

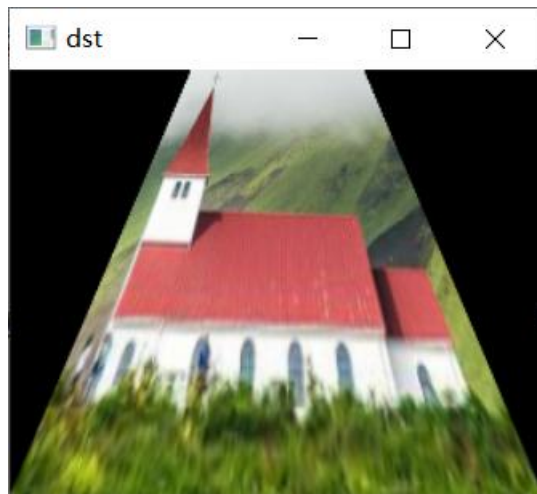
src：原图4个点坐标，格式为4行2列的32位浮点数列表，例如：[[0, 0], [1, 0], [0, 1],[1, 1]]。

dst：透视图的4个点坐标，格式与src一样。

返回值说明：

M：getPerspectiveTransform()方法计算出的仿射矩阵。

例 模拟从底部观察图像得到的透视效果。



## 例 模拟从底部观察图像得到的透视效果。

```
import cv2
import numpy as np
img = cv2.imread("t.png") # 读取图像
rows = len(img) # 图像像素行数
cols = len(img[0]) # 图像像素列数
p1 = np.zeros((4, 2), np.float32) # 32位浮点型空列表，保存原图四个点
p1[0] = [0, 0] # 左上角点坐标
p1[1] = [cols - 1, 0] # 右上角点坐标
p1[2] = [0, rows - 1] # 左下角点坐标
p1[3] = [cols - 1, rows - 1] # 右下角点坐标
p2 = np.zeros((4, 2), np.float32) # 32位浮点型空列表，保存透视图四个点
p2[0] = [90, 0] # 左上角点坐标，向右移动90像素
p2[1] = [cols - 90, 0] # 右上角点坐标，向左移动90像素
p2[2] = [0, rows - 1] # 左下角点坐标，位置不变
p2[3] = [cols - 1, rows - 1] # 右下角点坐标，位置不变
M = cv2.getPerspectiveTransform(p1, p2) # 根据四个点的变化轨迹计算出M矩阵
dst = cv2.warpPerspective(img, M, (cols, rows)) # 按照M进行仿射
cv2.imshow('img', img) # 显示原图
cv2.imshow('dst', dst) # 显示仿射变换效果
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 释放所有窗体
```

# 五 几何校正

图像在采集过程中很可能产生几何失真导致畸变，图像畸变的几何校正实际上一个图像恢复的过程，一般是以一幅图像为基准去校正另一幅图像的几何畸变。

# 五 几何校正

读入一幅图像，先对图像进行仿射变换，在进行逆变换对图像进行校正。（在对图像进行仿射变换时，图像的一部分内容会超出图像范围导致图像内容丢失，这时需要对图像进行扩充）

# 补充 图像扩充

```
cv2.copyMakeBorder(src,top,bottom,left,right,borderType[,value])
```

参数:

src输入图像

top,bottom,left,right上下左右对应的像素数目

borderType添加边界的类型

cv2.BORDER\_CONSTANT :添加有常数值边界，需要下一个参数value值。

cv2.BORDER\_REFLECT:以边缘为轴进行轴对称；比如: fedcba|abcdefgh|hgfedcb

cv2.BORDER\_REFLECT\_101: 以最边缘像素为轴进行轴对称；例如: gfedcb|abcdefgh|gfedcba

cv2.BORDER\_REPLICATE :重复最后一个元素；例如: aaaaaa|abcdefgh|hhhhhhh

cv2.BORDER\_WRAP :根据对边像素填充: cdefgh|abcdefgh|abcdefg

value 边界颜色，如果边界的类型是 cv2.BORDER\_CONSTANT

```
import cv2

img = cv2.imread('t.png')

img1=cv2.copyMakeBorder(img,50,50,50,50,cv2.BORDER_CONSTANT,value=(255,0,0))

img2=cv2.copyMakeBorder(img,50,50,50,50,cv2.BORDER_REFLECT)

img3=cv2.copyMakeBorder(img,50,50,50,50,cv2.BORDER_REFLECT_101)

img4=cv2.copyMakeBorder(img,50,50,50,50,cv2.BORDER_REPLICATE)

img5=cv2.copyMakeBorder(img,50,50,50,50,cv2.BORDER_WRAP)

cv2.imshow('new1',img1) #添加有常数值边界

cv2.imshow('new2',img2) #以边缘为轴进行轴对称

cv2.imshow('new3',img3) #以最边缘像素为轴进行轴对称;

cv2.imshow('new4',img4) #对边界像素进行复制;

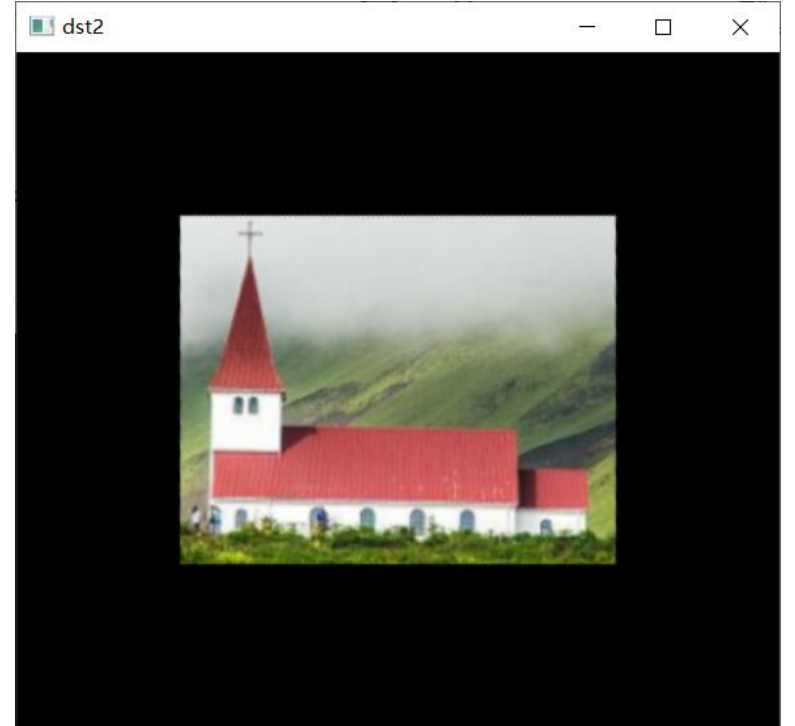
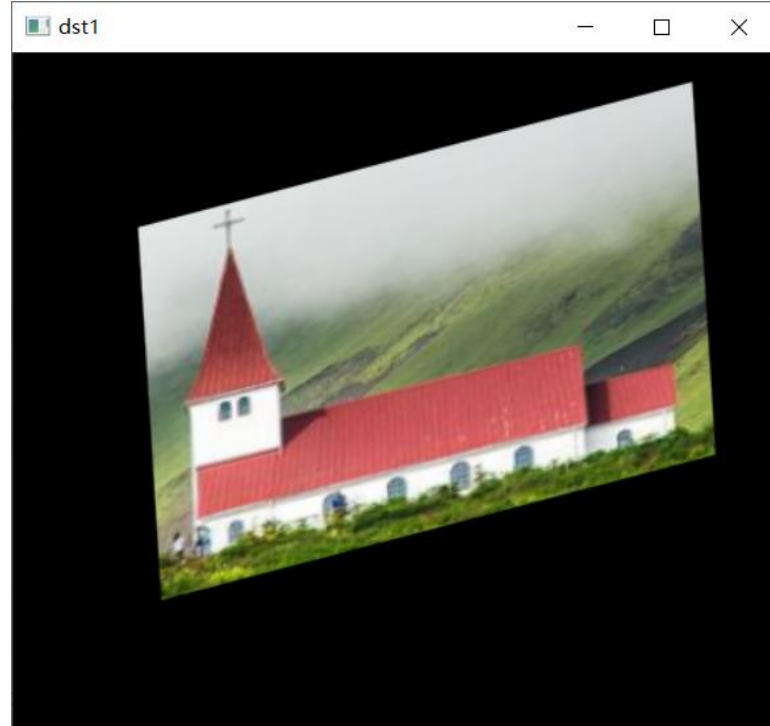
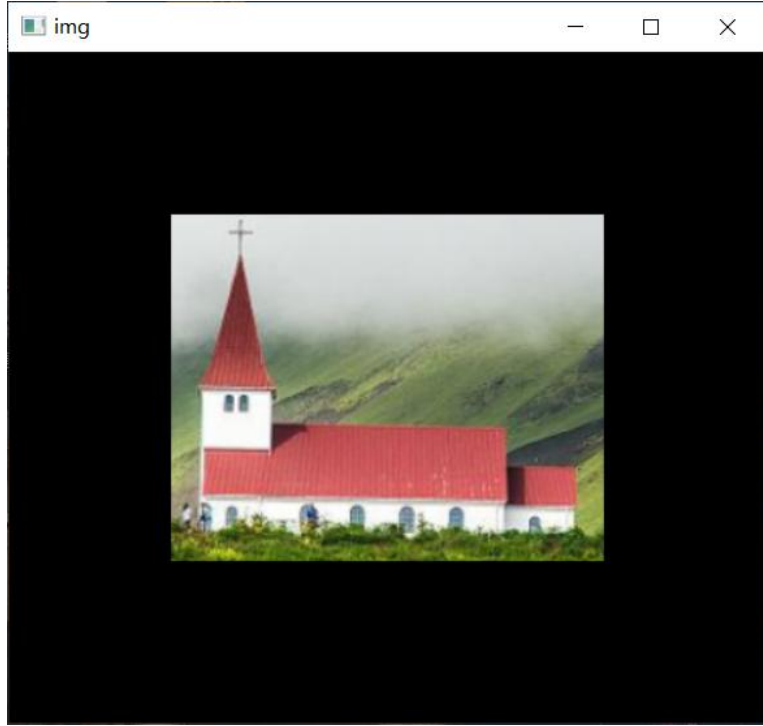
cv2.imshow('new5',img5) #根据对边像素填充;

cv2.waitKey() # 按下任何键盘按键后

cv2.destroyAllWindows() # 释放所有窗体
```



例：读入一幅图像，先对图像进行仿射变换，在进行逆变换对图像进行校正。



例：读入一幅图像，先对图像进行仿射变换，在进行逆变换对图像进行校正。

```
import cv2

import numpy as np

img = cv2.imread("t.png") # 读取图像
img=cv2.copyMakeBorder(img,100,100,100,100,cv2.BORDER_CONSTANT,value=(0,0,0))
rows,cols = img.shape[:2] # 图像像素行列数
p1 = np.float32([[50, 50],[100, 150],[200, 200]])
p2 = np.float32([[10, 70],[80, 160],[210, 180]])
M1 = cv2.getAffineTransform(p1, p2) # 根据三个点的变化轨迹计算出M矩阵
dst1 = cv2.warpAffine(img, M1, (cols, rows)) # 按照M进行仿射
M2 = cv2.getAffineTransform(p2, p1) # 逆仿射变换
dst2 = cv2.warpAffine(dst1, M2, (cols, rows)) # 仿射校正
cv2.imshow('img', img) # 显示原图
cv2.imshow('dst1', dst1) # 显示仿射变换效果
cv2.imshow('dst2', dst2) # 显示仿射校正效果
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 释放所有窗体
```

例：读入一幅图像，先对图像进行透视变换，在进行逆变换对图像进行校正。



例：读入一幅图像，先对图像进行透视变换，在进行逆变换对图像进行校正。

```
import cv2

import numpy as np

img = cv2.imread("t.png") # 读取图像
img=cv2.copyMakeBorder(img,100,100,100,100,cv2.BORDER_CONSTANT,value=(0,0,0))
rows,cols = img.shape[:2] # 图像像素行列数
p1 = np.float32([[0, 0],[cols - 1, 0],[0, rows - 1],[cols - 1, rows - 1]])
p2 = np.float32([[90, 0],[cols - 90, 0],[0, rows - 1],[cols - 1, rows - 1] ])
# M = cv2.getPerspectiveTransform(p1, p2) # 根据四个点的变化轨迹计算出M矩阵
M1 = cv2.getPerspectiveTransform(p1, p2)# 根据三个点的变化轨迹计算出M矩阵
dst1 = cv2.warpPerspective(img, M1, (cols, rows)) # 按照M进行仿射
M2 = cv2.getPerspectiveTransform(p2, p1) # 逆仿射变换
dst2 = cv2.warpPerspective(dst1, M2, (cols, rows)) # 仿射校正
cv2.imshow('img', img) # 显示原图
cv2.imshow('dst1', dst1) # 显示仿射变换效果
cv2.imshow('dst2', dst2) # 显示仿射校正效果
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 释放所有窗体
```

## 小结

- 1.图像的缩放：一种是设置dsize参数，另一种是设置fx参数和fy参数。
- 2.图像的翻转，沿X轴翻转、沿Y轴翻转和同时沿X轴、Y轴翻转，这3种方式均由flipCode参数的值决定。
- 3.图像的仿射变换：采用不同的仿射矩阵（M），就会使图像呈现不同的仿射效果。
- 4.图像的透视：依靠M矩阵实现。

 **THANKS** 