练习1 canny算子

```
Canny( InputArray dx, InputArray dy,
       OutputArray edges,
       double threshold1, double threshold2,
       bool L2gradient = false );
```

0. x方向梯度信息，(CV_16SC1 or CV_16SC3)。
1. y方向梯度信息，(CV_16SC1 or CV_16SC3)。
2. 输出，边缘检测结果
3. double类型的threshold1，第一个滞后性阈值。
4. double类型的threshold2，第二个滞后性阈值。
5. bool类型的L2gradient，一个计算图像梯度幅值的标识，有默认值false。

```
Canny( InputArray image, OutputArray edges,
       double threshold1, double threshold2,
       int apertureSize = 3, bool L2gradient = false );
```

0. 输入图像，即源图像，需为单通道8位图像。
1. 输出，边缘检测结果
2. double类型的threshold1，第一个滞后性阈值。
3. double类型的threshold2，第二个滞后性阈值。
4. 表示应用Sobel算子的核大小，默认值3。
5. bool类型的L2gradient，一个计算图像梯度幅值的标识，有默认值false。

练习1 canny算子

重载函数是函数的一种特殊情况，为方便使用，C++允许在同一范围中声明几个功能类似的同名函数，但是这些同名函数的形式参数（指参数的个数、类型或者顺序）必须不同，也就是说用同一个函数完成不同的功能。这就是重载函数。重载函数常用来实现功能类似而所处理的数据类型不同的问题。不能只有函数返回值类型不同。

练习2 旋转及缩放

根据角度及缩放比例
生成仿射变换矩阵

仿射变换函数

```
cv::Mat dstMat;
cv::Mat srcMat = cv::imread("D:\\lena.jpg", 1);
if (srcMat.empty()) return -1;

//旋转-40°，缩放尺度为
float angle = -10.0, scale = 1;
// 旋转中心为图像中心
cv::Point2f center(srcMat.cols*0.5, srcMat.rows*0.5);
//获得变换矩阵
const cv::Mat affine_matrix = cv::getRotationMatrix2D(center, angle, scale);

cv::warpAffine(srcMat, dstMat, affine_matrix, srcMat.size());

cv::imshow("src", srcMat);
cv::imshow("dst", dstMat);
cv::waitKey(0);
```

练习2 旋转及缩放

根据角度及缩放比例
生成仿射变换矩阵

仿射变换函数

```
cv::Mat dstMat;
cv::Mat srcMat = cv::imread("D:\\lena.jpg", 1);
if (srcMat.empty()) return -1;

//旋转-40°，缩放尺度为
float angle = -10.0, scale = 1;
// 旋转中心为图像中心
cv::Point2f center(srcMat.cols*0.5, srcMat.rows*0.5);
//获得变换矩阵
const cv::Mat affine_matrix = cv::getRotationMatrix2D(center, angle, scale);

cv::warpAffine(srcMat, dstMat, affine_matrix, srcMat.size());

cv::imshow("src", srcMat);
cv::imshow("dst", dstMat);
cv::waitKey(0);
```

练习3 仿射变换

变换前的三点坐标

变换后的三点坐标

根据角度及缩放比例
生成仿射变换矩阵

仿射变换函数

```
cv::Mat dstMat;
cv::Mat srcMat = cv::imread("D:\\lena.jpg", 1);
if (srcMat.empty()) return -1;

//变换前的三点坐标
const cv::Point2f src_pt[] = {
                                cv::Point2f(200, 200),
                                cv::Point2f(250, 200),
                                cv::Point2f(200, 100) };
//变换后的三点坐标
const cv::Point2f dst_pt[] = {
                                cv::Point2f(300, 100),
                                cv::Point2f(300, 50),
                                cv::Point2f(200, 100) };
//计算仿射矩阵
const cv::Mat affine_matrix = cv::getAffineTransform(src_pt, dst_pt);

cv::warpAffine(srcMat, dstMat, affine_matrix, srcMat.size());

cv::imshow("src", srcMat);
cv::imshow("dst", dstMat);
cv::waitKey(0);
```

练习4 投影变换

变换前的四点坐标

变换后的四点坐标

根据角度及缩放比例
生成仿射变换矩阵

仿射变换函数

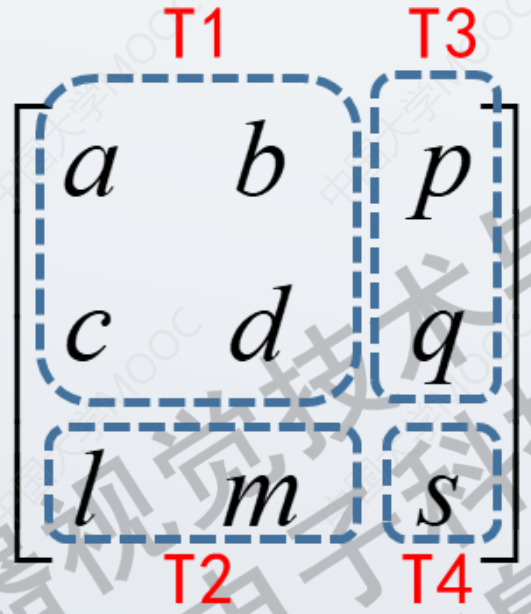
```
cv::Mat dstMat;
cv::Mat srcMat = cv::imread("D:\\lena.jpg", 1);
if (srcMat.empty()) return -1;

//变换前的四点坐标
cv::Point2f pts1[] = {
                        cv::Point2f(150,150),
                        cv::Point2f(150,300),
                        cv::Point2f(350,300),
                        cv::Point2f(350,150) };
//变换后的四点坐标
cv::Point2f pts2[] = {
                        cv::Point2f(200,150),
                        cv::Point2f(200,300),
                        cv::Point2f(340,270),
                        cv::Point2f(340,180) };

// 透視変換行列を計算
cv::Mat perspective_matrix = cv::getPerspectiveTransform(pts1, pts2);
// 变换
cv::warpPerspective(srcMat, dstMat, perspective_matrix, srcMat.size());

cv::imshow("src", srcMat);
cv::imshow("dst", dstMat);
cv::waitKey(0);
```

仿射矩阵



T1：比例、旋转、对称、错切

T2：平移

T3：投影

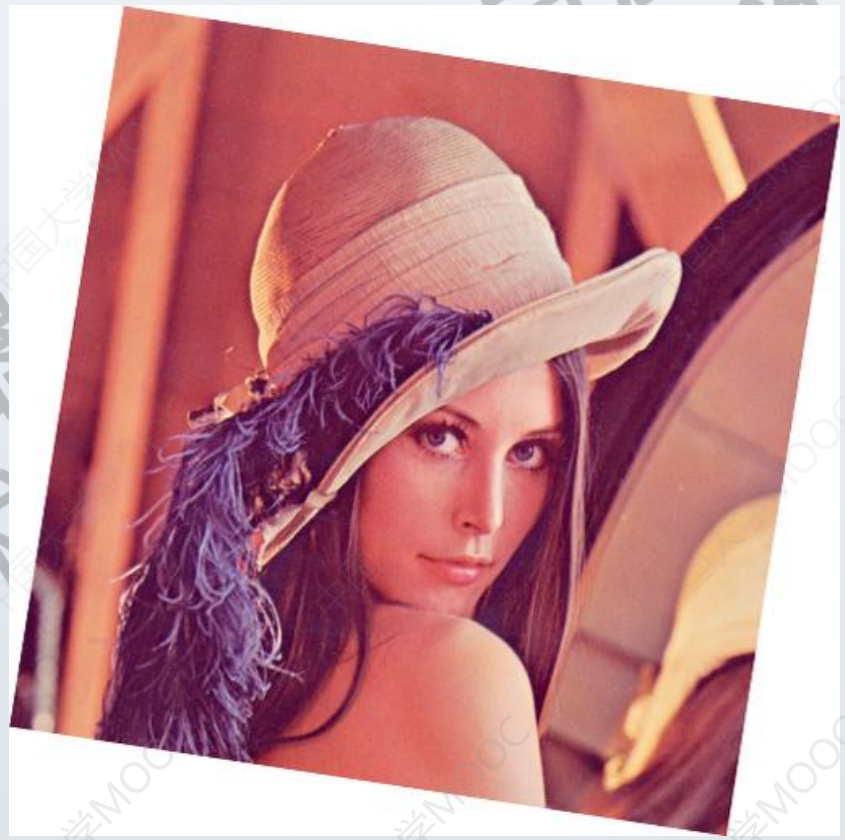
T4：整体缩放

OpenCV中为3x2矩阵

$$\begin{bmatrix} a & c & l \\ b & d & m \end{bmatrix}$$

练习5 图像矫正

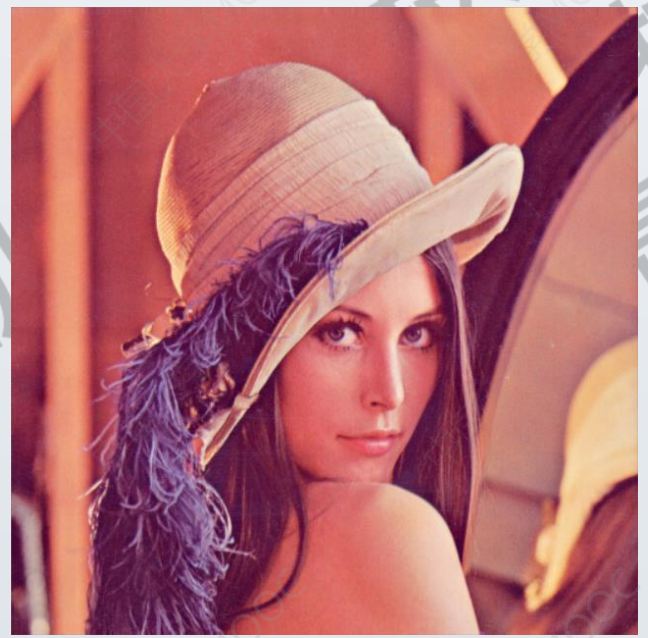
现有如下图像，如何自动矫正成尺寸相同的左图。



思考

通过OpenCV进行图像的旋转后，超出原尺寸的部分，会被自动裁剪，如何实现不自动裁剪的图像旋转

原图



原旋转

改进后的旋转