第 5 章

图像的几何变换

图像的几何变换是指将一幅图像中的坐标位置映射到另一张图像中的新坐标位置，几何变换包括平移、旋转、缩放、翻转、仿射变换、透视变换等。平移、旋转、缩放似乎是最为简单的几何变换，但是 OpenCV 中并没有专门用于平移和旋转的函数，而要通过仿射变换实现，因此，介绍图像的几何变换需要从仿射变换开始。

5.1 仿射变换

仿射变换是将一个二维坐标转换到另一个二维坐标的过程。仿射变换是一种线性变换，变换前是直线的，变换后依然是直线；变换前是平行线的，变换后依然是平行线。

仿射变换的概念如图 5-1 所示。变换前图像中的点 1、点 2、点 3（不在同一条直线上）与变换后图像中的点 1、点 2、点 3 一一对应。由于 3 点可以决定一个平面，所以利用这 3 个点的对应关系就可以对整个图像平面进行仿射变换。

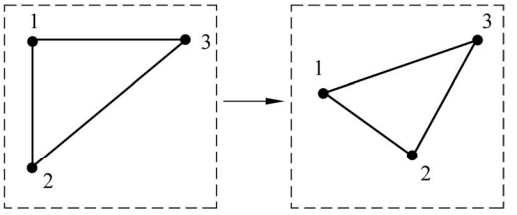


图 5-1　仿射变换示意图

在 OpenCV 中实现仿射变换需要两步，第 1 步通过 getAffineTransform()函数来计算仿射变换矩阵（矩阵大小为 2×3），第 2 步通过 warpAffine()函数实现仿射变换。

OpenCV 中用于计算仿射变换矩阵的函数原型如下：

```
Mat Imgproc.getAffineTransform(MatOfPoint2f src, MatOfPoint2f dst)
函数用途：计算仿射变换矩阵。

【参数说明】
(1) src：原图像中的 3 个点的坐标。
(2) dst：原图像中 3 个点在目标图像中对应的坐标。
```

OpenCV 中用于实现仿射变换的函数原型如下：

```
void Imgproc.warpAffine(Mat src, Mat dst, Mat M, Size dsize, int flags);
```

函数用途：对图像进行仿射变换。

【参数说明】

(1) src：输入图像。

(2) dst：输出图像，尺寸和 dsize 一致，数据类型与 src 相同。

(3) dsize：输出图像的尺寸。

(4) flags：差值方法，常用参数如下。

◆ Imgproc.INTER_NEAREST：最近邻插值。

◆ Imgproc.INTER_LINEAR：线性插值。

◆ Imgproc.INTER_AREA：区域插值。

◆ Imgproc.INTER_CUBIC：三次样条插值。

◆ Imgproc.INTER_LANCZOS4：Lanczos 差值。

◆ Imgproc.INTER_LINEAR_EXACT：位精确双线性插值。

◆ Imgproc.INTER_NEAREST_EXACT：位精确最近邻插值。

◆ Imgproc.INTER_MAX：用掩码进行插值。

◆ Imgproc.WARP_FILL_OUTLIERS：填充所有输出图像像素，如有像素落在输入图像边界外，则将它们设为 0。

◆ Imgproc.WARP_INVERSE_MAP：反变换。

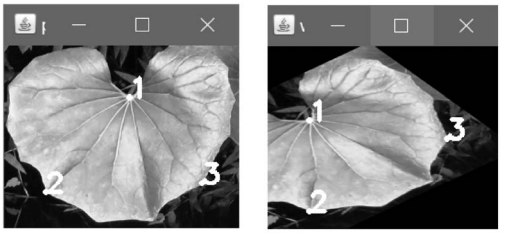
仿射变换的范围很广，平移、旋转、缩放、翻转实际上都属于仿射变换。

下面用一个完整的程序说明仿射变换的过程，代码如下：

```
//第 5 章/WarpAffine.java

import org.opencv.core.*;
import org.opencv.highgui.HighGui;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.imgproc.Imgproc;

public class WarpAffine {

          public static void main(String[] args) {
                    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

                    //读取图像
                    Mat src=Imgcodecs.imread("leaf.png");

                    //定义原图像中 3 个点的坐标
                    Point[] pt1 = new Point[3];
                    pt1[0] = new Point(136, 56);
                    pt1[1] = new Point(45, 160);
                    pt1[2] = new Point(215, 150);

                    //定义目标图像中 3 个点的坐标
```

```
                        Point[] pt2 = new Point[3];    //输出图像点集
                        pt2[0] = new Point(50, 80);
                        pt2[1] = new Point(50, 180);
                        pt2[2] = new Point(200, 100);

                        //getAffineTransform()函数用到的数据类型
                        MatOfPoint2f mop1 = new MatOfPoint2f(pt1);
                        MatOfPoint2f mop2 = new MatOfPoint2f(pt2);

                        //计算仿射变换矩阵并进行仿射变换
                        Mat dst=new Mat();
                        Mat mat=Imgproc.getAffineTransform(mop1, mop2);
                        Imgproc.warpAffine(src, dst, mat, src.size());

                        //在原图像中标记 3 个点的坐标
                        Scalar color=new Scalar(255,255,255);
                        for (int n=0; n<3; n++) {
                                  Imgproc.circle(src, pt1[n], 2, color, 2);
                                  Imgproc.putText(src, n + 1 + "", pt1[n], 0, 1,
color,3);
                        }

                        //在目标图像中标记 3 个点的坐标
                        for (int n=0; n<3; n++) {
                                  Imgproc.circle(dst, pt2[n], 2, color, 2);
                                  Imgproc.putText(dst, n + 1 + "", pt2[n] , 0 ,
1, color,3);
                        }

                        //在屏幕上显示变换前后的图像
                        HighGui.imshow("src", src);
                        HighGui.waitKey(0);
                        HighGui.imshow("warped", dst);
                        HighGui.waitKey(0);

                        //在控制台输出仿射变换矩阵
                        System.out.println(mat.dump());
                        System.exit(0);

              }
    }
```

程序的运行结果如图 5-2 所示，图中用白色字体标出了变换前后 3 个定位点的位置。此外，程序最后还在控制台输出了仿射转换矩阵供参考，如图 5-3 所示。这个矩阵看上去再简单不过了，但就是这样简单的矩阵却能够轻松地实现各种功能。在后续章节中还会有更多的矩阵能够实现各种眼花缭乱的功能。



(a) 原图像　　(b) 仿射变换后

图 5-2　WarpAffine.java 程序的运行结果

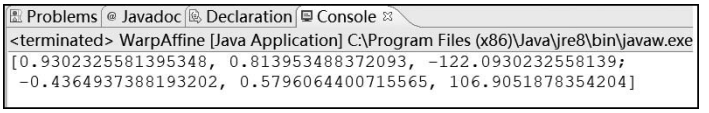


图 5-3　WarpAffine.java 程序中的仿射变换矩阵

5.2　透视变换

仿射变换又称三点变换，因为它只用到 3 个点，而透视变换则用到了 4 个点，因此也被称为四点变换。透视变换是利用投影成像的原理将物体重新投射到另一个成像平面，如图 5-4 所示。透视变换的转换矩阵也与仿射变换的矩阵不同，是一个 3×3 的矩阵。

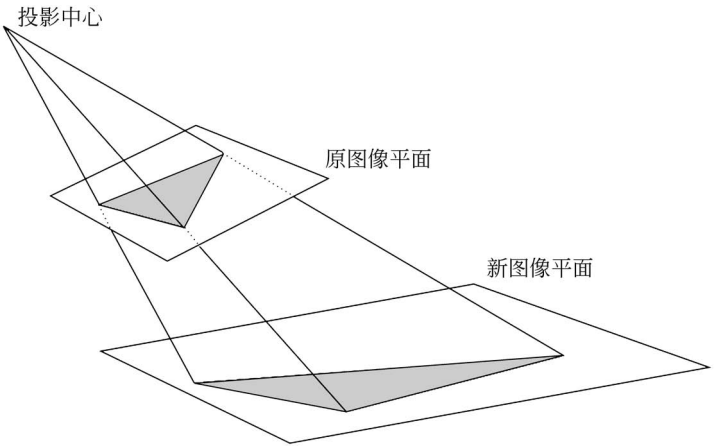


图 5-4　透视变换示意图

在 OpenCV 中实现透视变换也分两步，第 1 步通过 getPerspectiveTransform()函数来计算透视变换矩阵（3×3），第 2 步通过 warpPerspective()函数实现透视变换。

OpenCV 中用于计算透视变换矩阵的函数原型如下：

```
Mat Imgproc.getPerspectiveTransform(Mat src, Mat dst)
```
函数用途：根据 4 对对应的点计算透视变换的矩阵。

【参数说明】
(1) src：原图像中的 4 个点的坐标。
(2) dst：原图像中 4 个点在目标图像中对应的坐标。

OpenCV 中用于实现透视变换的函数原型如下：

```
void Imgproc.warpPerspective(Mat src, Mat dst, Mat M, Size dsize)
```
函数用途：对图像进行透视变换。

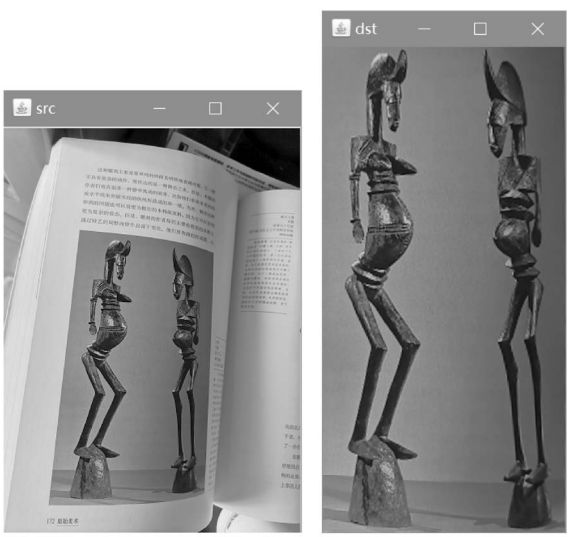
【参数说明】
(1) src：输入图像。
(2) dst：输出图像，尺寸和 dsize 一致，数据类型与 src 相同。
(3) M：3*3 的变换矩阵。
(4) dsize：输出图像的尺寸。

下面用一个完整的程序说明如何实现透视变换，代码如下：

```
//第 5 章/PerspectiveTransform.java

import org.opencv.core.*;
import org.opencv.highgui.HighGui;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.imgproc.Imgproc;

public class PerspectiveTransform  {

          public static void main(String[] args) {
                    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

                    //读取图像
                    Mat src=Imgcodecs.imread("book.png");

                    //定义原图像中 4 个点的坐标
                    Point[] pt1 = new Point[4];
                    pt1[0] = new Point(95, 129);
                    pt1[1] = new Point(260, 157);
                    pt1[2] = new Point(57, 469);
                    pt1[3] = new Point(248, 454);

                    //定义目标图像中 4 个点的坐标
                    Point[] pt2 = new Point[4];
                    pt2[0] = new Point(0, 0);
                    pt2[1] = new Point(300, 0);
                    pt2[2] = new Point(0,600);
                    pt2[3] = new Point(300,600);

                    //getPerspectiveTransform()函数用到的数据类型
                    MatOfPoint2f mop1 = new MatOfPoint2f(pt1);
```

```
                        MatOfPoint2f mop2 = new MatOfPoint2f(pt2);

                        //计算透视变换矩阵并进行仿射变换
                        Mat dst=new Mat();
                        Mat matrix=Imgproc.getPerspectiveTransform(mop1, mop2);
                        //获取转换矩阵
                        Imgproc.warpPerspective(src, dst, matrix, new
Size(300,600));

                        //在屏幕上显示变换前后的图像
                        HighGui.imshow("src", src);
                        HighGui.waitKey(0);
                        HighGui.imshow("dst", dst);
                        HighGui.waitKey(0);
                        System.exit(0);
                }

    }
```

程序的运行结果如图 5-5 所示。程序的输入图像是书中的一页，其 4 个定位点明显不在一个平面上，经过透视变换以后还原成一张完整的图像。



(a) 原图像　　(b) 透视变换后

图 5-5　PerspectiveTransform.java 程序的运行结果

5.3　平移

图像平移是将一张图像中所有的点都按照指定的平移量在水平和垂直方向上进行移动，平移后的图像与原图像相同。平移后图像上的每个点都可以在原图像中找到对应的点。假设

原图像的点的坐标为(x_0, y_0)，平移量为$(\Delta x, \Delta y)$，平移后坐标为(x_1, y_1)，平移前后的坐标关系可以用数学公式表示如下：

$$x_1=x_0+\Delta x \tag{5-1}$$
$$y_1=y_0+\Delta y$$

平移的实现有多种方法，最容易想到的方法就是利用循环语句对像素赋值。这种方法其实并不需要 OpenCV 就可以实现，代码从略。那么 OpenCV 中有没有函数可以实现平移呢？答案是有的，利用仿射变换的 warpAffine()函数同样可以实现图像平移。

实际上，运用 5.1 节的知识已经可以写出这个程序。仿射变换的关键是找到三组对应点，有了这些点就能计算出仿射变换的矩阵。现在定义 3 个点，例如(0,0)、(0,10)和(10,0)。假设 x 方向平移 30 像素，y 方向平移 50 像素。简单计算可知平移后这 3 个点的坐标为(30,50)、(30,60)和(40,50)，然后将 WarpAffine.java 程序中的坐标用这些点的坐标替换即可求出相应的转换矩阵并进行平移操作。

但是这个过程还是有点烦琐。实际上，平移这种简单操作的转换矩阵也很简单。让图像在 x 方向平移 Δx，在 y 方向平移 Δy 的矩阵如下：

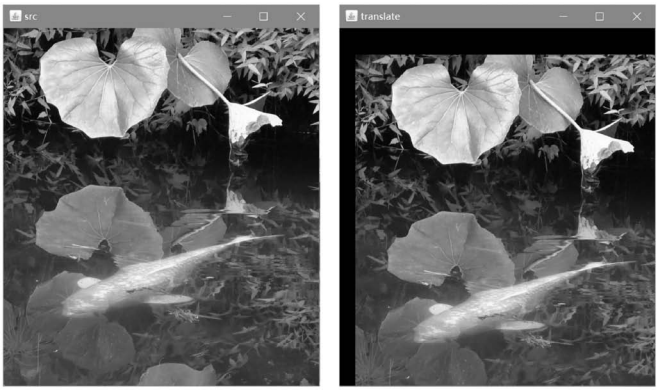
$$\begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \end{bmatrix}$$

知道了这一点，用仿射变换函数实现平移也就简单多了。下面用一个完整的程序说明如何用仿射变换函数实现平移操作，代码如下：

```
//第 5 章/Translate.java

import org.opencv.core.*;
import org.opencv.highgui.HighGui;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.imgproc.Imgproc;

public class Translate {

          public static void main(String[] args) {
                    System.loadLibrary( Core.NATIVE_LIBRARY_NAME );

                    //构建用于平移的矩阵
                    Mat mat = new Mat(2, 3, CvType.CV_64F);
                    double [] Data=new double[] {1,0,30,0,1,50};
                    mat.put(0, 0, Data);

                    //读取图像并在屏幕上显示
                    Mat src=Imgcodecs.imread("pond.png");
                    HighGui.imshow("src", src);
                    HighGui.waitKey(0);

                    //进行平移并在屏幕上显示平移后的图像
                    Mat dst=new Mat();
```

```
                    Imgproc.warpAffine(src, dst, mat, src.size());
                    HighGui.imshow("translate", dst);
                    HighGui.waitKey(0);
                    System.exit(0);

            }

    }
```

程序的运行结果如图 5-6 所示。程序中利用 Mat 类的 put()函数构建了平移矩阵，然后用 warpAffine()函数实现了平移操作。需要注意平移矩阵的数据类型。



(a) 原图像　　(b) 平移后

图 5-6　Translate.java 程序的运行结果

5.4　旋转

在 OpenCV 中也没有专门用于旋转的函数，实现图像的旋转同样是通过仿射变换实现的。在 OpenCV 中实现图像的旋转分为两步，第 1 步通过 getRotationMatrix2D()函数来计算旋转矩阵，第 2 步通过 warpAffine()函数实现旋转。

计算旋转矩阵的函数原型如下：

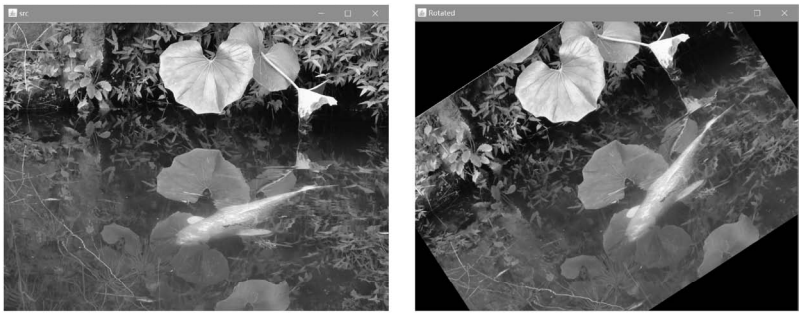
```
Mat Imgproc.getRotationMatrix2D(Point center, double angle, double scale)
函数用途：计算二维旋转的仿射矩阵。

【参数说明】
(1) center：原图像中的旋转中心。
(2) angle：以度为单位的旋转角度，正值表示逆时针旋转（假设坐标原点为左上角）。
(3) scale：缩放比例因子。旋转中可以实现缩放，如果不缩放，则用 1 表示。
```

下面用一个完整的程序说明如何实现图像的旋转，代码如下：

```
//第 5 章/Rotate.java
```

```
import org.opencv.core.*;
import org.opencv.highgui.HighGui;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.imgproc.Imgproc;

public class Rotate {

        public static void main(String[] args) {
                System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

                //读入图像并在屏幕上显示
                Mat src=Imgcodecs.imread("fish.png");
                HighGui.imshow("src", src);
                HighGui.waitKey(0);

                //计算旋转用的仿射矩阵
                Mat dst=new Mat();
                Point center =new Point(src.width()/2.0, src.height()/2.0);
                Mat matrix =Imgproc.getRotationMatrix2D(center, 33.0, 1.0);

                //旋转图像并在屏幕上显示
                Imgproc.warpAffine(src, dst, matrix, src.size(),
Imgproc.INTER_LINEAR);
                HighGui.imshow("Rotated", dst);
                HighGui.waitKey(0);
                System.exit(0);
        }
}
```

程序的运行结果如图 5-7 所示。本例中将旋转中心设为图像的中心位置，旋转角度为33°，正数表示逆时针旋转，所以图像逆时针旋转了 33°。



(a) 原图像　　(b) 旋转后

图 5-7　Rotate.java 程序的运行结果

5.5　缩放

图像的缩放就是改变图像的尺寸。OpenCV 中用于改变图像尺寸的函数原型如下：

```
    void Imgproc.resize(Mat src, Mat dst, Size dsize, double fx, double fy)
    函数用途：改变图像的大小。

    【参数说明】
    (1) src：输入图像。
    (2) dst：输出图像，其数据类型与 src 相同。
    (3) dsize：输出图像的大小。如 dsize 的大小为 0，则 dsize=Size(round(fx*src.cols),
round(fy*src.rows))。
    (4) fx：水平方向缩放比例。如 fx 为 0，则 fx=(double)dsize.width/src.cols。
    (5) fy：垂直方向缩放比例。如 fy 为 0，则 fy=(double)dsize.height/src.rows。
```

此函数实际上有两套参数，可以通过 dsize 设置输出图像大小或者通过 fx 和 fy 设置缩放比例。两套参数使用哪一套应根据参数值确定，详见参数说明。

下面用一个完整的程序说明图像缩放的方法，代码如下：

```
    //第 5 章/Scale.java

    import org.opencv.core.*;
    import org.opencv.highgui.HighGui;
    import org.opencv.imgcodecs.Imgcodecs;
    import org.opencv.imgproc.Imgproc;

    public class Scale {

        public static void main(String[] args) {
                        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

                        //读入图像并在屏幕上显示
                        Mat src=Imgcodecs.imread("church.png");
                        HighGui.imshow("src", src);
                        HighGui.waitKey(0);

                        //获取原图像尺寸
                        float width=src.width();
                        float height=src.height();

                        //将原图像放大至 1.2 倍
                        Mat dst1=new Mat();
                        float scale1=1.2f;  //缩放比例 1
                        Imgproc.resize(src, dst1, new Size(width*scale1,
height*scale1));
```