第 3 章　图像的几何变换

本章学习目标：

(1) 了解数字图像的坐标系统。

(2) 了解图像几何变换中向前映射和向后映射的基本原理及适用场景。

(3) 掌握最邻近插值法、双线性插值法和双三次插值法的插值原理。

(4) 掌握平移、镜像、旋转、缩放、错切、转置等仿射变换和透视变换的几何变换矩阵表示及实现方法。

图像的几何变换是在不改变图像内容的前提下对图像像素进行相对空间位置移动，从而重构图像的空间结构，达到处理图像的目的。根据变换的性质可以划分为仿射变换和透视变换。其中，仿射变换包括平移、镜像、旋转、缩放、错切基本变换和转置复合变换。

3.1　图像几何变换的理论基础

3.1.1　坐标系统

1. 像素坐标系

MATLAB 将读入的图像存储为二维数组即矩阵，其中矩阵中的每个元素对应于该图像中的单个像素，因此通常表示像素在图像中位置最简单的方法是像素坐标。对于像素坐标，第一个分量 r(行)向下增长，第二个分量 c(列)向右增长，且像素坐标是整型数值，其数据范围在 1 到行数或列数之间，如图 3-1 所示。

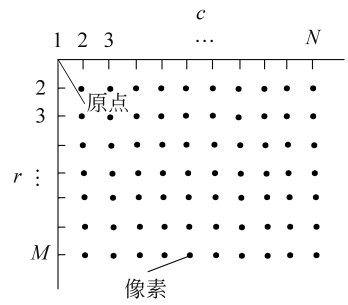


图 3-1　像素坐标系

在像素坐标系中，一个像素被认为是一个离散单元，由一个单独的坐标对唯一决定，例如(3,7)，而(3.2,7.6)这样的位置坐标是没有意义的。并且无论是单通道图像还是多通道图像都可以使用前两个矩阵维度的像素坐标来表示单个像素在图像中的位置，例如 $A(5,3)$、$B(4,6,:)$。

2. 空间坐标系

除了离散的像素坐标方法，还可以使用连续变化的空间坐标系来表示像素在图像中的位置。在 MATLAB 图像处理工具箱中定义了两种类型的空间坐标系：内部坐标系和世界坐标系，默认情况下使用的是内部坐标系。

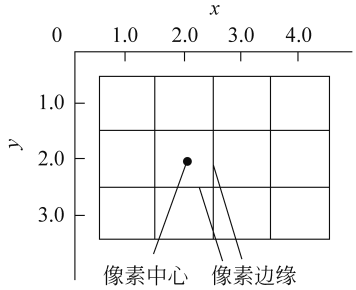


图 3-2　内部坐标系

1) 内部坐标系

内部坐标系是与像素坐标系一致的空间坐标系，如图 3-2 所示。在这种坐标系中，x 轴是水平的，并向右延伸，y 轴是垂直的，并向下延伸，每个像素中心的内部坐标都是整型数值。

由于连续的内部坐标系中每个像素的大小是一个单位，因此其边缘具有小数坐标。从这个角度看，坐标(3.2,5.3)是有意义的，并且不

同于像素坐标(5,3)。

注意：任何像素中心的内部坐标(x,y)与该像素的像素坐标(r,c)之间的关联是横坐标x与其列索引c相同，纵坐标y与其行索引r相同，即像素坐标系中第5行第3列的像素位于内部坐标系中$x=3$、$y=5$的空间坐标位置。

2）世界坐标系

由于摄像机可安放在任意位置，在环境中选择一个基准坐标系来描述摄像机的位置，并用它描述环境中任何物体的位置，此绝对坐标系称为世界坐标系。在某些情况下，可能希望使用世界坐标系进行描述。例如，当对图像执行几何变换并希望保留新位置与原始位置的关系时，就需要从内部坐标系转换到世界坐标系。

3.1.2　图像几何变换概述

图像几何变换用于改变图像中像素与像素之间的空间关系，从而重构图像的空间结构，达到处理图像的目的。简而言之，图像几何变换就是建立一种输入图像与变换后输出图像对应像素坐标之间的映射关系。其数学公式描述如下：

$$\begin{cases} x = U(x_0, y_0) \\ y = V(x_0, y_0) \end{cases} \tag{3-1}$$

其中，(x,y)表示输出图像像素的坐标，(x_0,y_0)表示输入图像像素的坐标，U、V表示两种映射函数。它可以是线性关系，也可以是非线性关系，即

$$\begin{cases} U(x,y) = k_1 x + k_2 y + k_3 \\ V(x,y) = k_4 x + k_5 y + k_6 \end{cases} \tag{3-2}$$

$$\begin{cases} U(x,y) = k_1 + k_2 x + k_3 y + k_4 x^2 + k_5 xy + k_6 y^2 \\ V(x,y) = k_7 + k_8 x + k_9 y + k_{10} x^2 + k_{11} xy + k_{12} y^2 \end{cases} \tag{3-3}$$

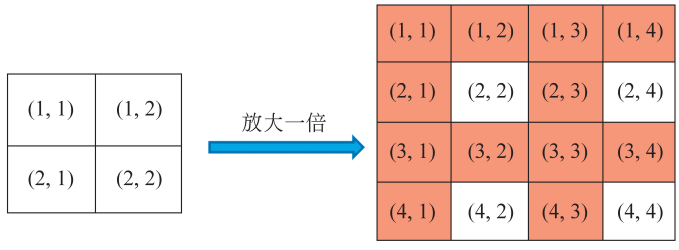
可以看出，只要给定输入图像上任意像素的坐标，都能够通过以上映射关系获得几何变换后的输出图像对应像素的坐标，这种将输入映射到输出的过程称为向前映射。但在向前映射中，多个输入坐标可对应同一个输出坐标，由此会带来以下问题。

1. 产生无效的浮点数坐标

输入图像中非负整数所表示的像素坐标通过映射函数变换后可能会得到浮点数的坐标，如输入图像像素坐标(1,1)在缩小一半后对应的输出像素坐标为(0.5,0.5)。

2. 映射不完全

当输入图像的像素总数小于输出图像的像素总数时，例如放大、旋转变换，会导致输出图像的部分像素(图3-3中红色标注的像素)在输入图像中不存在映射关系。这种“映射不完全”会产生有规律的空洞(黑色蜂窝状)，如图3-4所示。



(a) 输入图像　　(b) 输出图像

图3-3　“映射不完全”示意图

(a) 放大一倍后的映射效果

(b) 顺时针旋转 30°后的映射效果

图 3-4 “映射不完全”效果

3. 映射重叠

当输入图像的像素总数大于输出图像的像素总数，例如缩小变换时，会导致输入图像中多个像素坐标对应于输出图像的同一个坐标，如图 3-5(a)所示输入图像中红色标注的像素坐标(1,1)、(1,2)、(2,1)、(2,2)经过缩小一半后在输出图像中对应的像素坐标为(0.5,0.5)、(0.5,1)、(1,0.5)、(1,1)，经四舍五入后均为(1,1)，产生了“映射重叠”的现象。

(1, 1)	(1, 2)	(1, 3)	(1, 4)
(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 1)	(3, 2)	(3, 3)	(3, 4)
(4, 1)	(4, 2)	(4, 3)	(4, 4)

(a) 输入图像

缩小一倍

(1, 1)	(1, 2)
(2, 1)	(2, 2)

(b) 输出图像

图 3-5 “映射重叠”示意图

为了解决“向前映射”的映射不完全和映射重叠问题，引入了另一种映射方法“向后映射”。其数学表达描述为

$$\begin{cases} x_0 = U'(x, y) \\ y_0 = V'(x, y) \end{cases} \tag{3-4}$$

与“向前映射”相反，它是由输出图像像素坐标(x, y)反过来推算出该像素在输入图像中的坐标(x_0, y_0)。正因为输出图像的每个像素坐标都能通过这个映射关系找到输入图像中对应的坐标位置，因此不会出现“映射不完全”和“映射重叠”的问题。

综上所述，“向前映射”适用于不改变图像大小的几何变换，如平移、镜像，而“向后映射”则适用于改变图像大小的旋转、缩放、错切和透视变换。但无论是“向前映射”还是“向后映射”，均会产生无效的浮

点数坐标，这时就需要借助插值算法获得此浮点数坐标像素的近似值。

3.1.3　图像插值算法

常见的图像插值算法有最近邻插值法，双线性插值法和双三次插值法。这里以“放大”几何变换为例，采用“向后映射”方式，对这 3 种插值算法的基本思想进行详尽阐述。

1. 最近邻插值法

最近邻插值法的基本思想是将浮点数坐标像素的值设置为距离该像素最近的输入图像像素的值，简言之，就是“四舍五入”。

假设现有 3×3 大小的灰度输入图像，其图像矩阵如下：

$$\begin{bmatrix} 50 & 120 & 98 \\ 210 & 45 & 12 \\ 180 & 68 & 112 \end{bmatrix}$$

输出图像期望大小为 5×5，则所构建输出图像与输入图像之间的坐标对应关系如下：

$$\begin{cases} x_0 = \dfrac{3}{5}x \\ y_0 = \dfrac{3}{5}y \end{cases} \tag{3-5}$$

首先，输出图像中第一行像素的坐标(1,1)、(1,2)、(1,3)、(1,4)、(1,5)通过式(3-5)所得到的变换后坐标为(0.6,0.6)、(0.6,1.2)、(0.6,1.8)、(0.6,2.4)、(0.6,3)，最近邻插值法是将这些浮点数坐标通过四舍五入运算来获得整数坐标(1,1)、(1,1)、(1,2)、(1,2)、(1,3)，如图 3-6 所示。

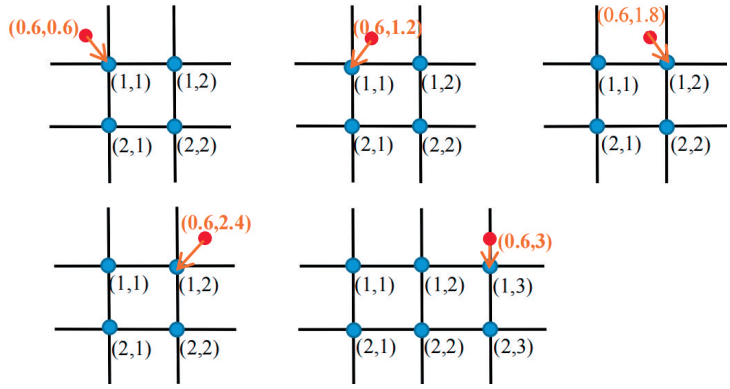


图 3-6　最近邻插值法的近似坐标计算结果

其次，将输入图像中坐标(1,1)、(1,2)、(1,3)的像素值 50、120、98 分别映射到输出图像中对应坐标(1,1)、(1,2)、(1,3)、(1,4)、(1,5)的像素上。全部求解完毕后得到的输出图像矩阵如下：

$$\begin{bmatrix} 50 & 50 & 120 & 120 & 98 \\ 50 & 50 & 120 & 120 & 98 \\ 210 & 210 & 45 & 45 & 12 \\ 210 & 210 & 45 & 45 & 12 \\ 180 & 180 & 68 & 68 & 112 \end{bmatrix}$$

通过上述示例可以看出，最近邻插值法仅需做“四舍五入”的取整运算，其计算量较小，因而速度相当快。但会出现因输出图像中方块区域内的数值相等而造成大量“马赛克”或“锯齿”现象，如图 3-7 所示。

2. 双线性插值法

双线性插值法的基本思想是在 x 和 y 两个方向分别进行一次线性插值，即输出图像中每个像素的值都是由输入图像中相对应像素的相邻 4 个像素值的加权平均得到的。

例如，将上述示例中输出图像的像素坐标(3,4)代入式(3-5)计算得到输入图像中对应坐标为(1.8,2.4)。双线性插值法的具体计算过程如下。

步骤 1：过坐标(1.8,2.4)在输入图像上作垂线与上下两条水平线相交于坐标(1,2.4)、(2,2.4)，如图 3-8 所示。

图 3-7 “马赛克”现象

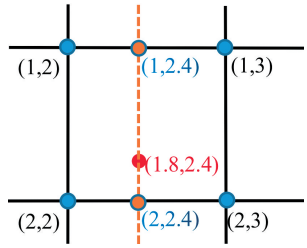


图 3-8 双线性插值法原理示意图

步骤 2：x 方向上的线性插值。对于浮点数坐标(1,2.4)而言，与之距离最近的两个像素坐标分别是(1,2)、(1,3)，且相距的距离分别为 0.4 和 0.6。遵循距离越近、贡献越大的原则，这里将坐标(1,2)的贡献值即权值设置为 0.6，而坐标(1,3)的贡献值即权值设置为 0.4，对其加权平均的计算公式如下：

$$T_{(1,2.4)}=0.6\times T_{(1,2)}+0.4\times T_{(1,3)}=0.24 \tag{3-6}$$

其中，$T_{(i,j)}$ 代表坐标(i,j)对应像素的值。同理，对于浮点数坐标(2,2.4)，对其加权平均的计算公式如下：

$$T_{(2,2.4)}=0.6\times T_{(2,2)}+0.4\times T_{(2,3)}=0.54 \tag{3-7}$$

步骤 3：y 方向上的线性插值。将步骤 2 所得到的 $T_{(1,2.4)}$、$T_{(2,2.4)}$ 再在 y 方向上进行一次线性插值即可得到 $T_{(1.8,2.4)}$。这里将与坐标(1.8,2.4)相距较远的坐标(1,2.4)权值设置为 0.2，相距较近的坐标(2,2.4)的权值设置为 0.8，对其加权平均的计算公式如下：

$$T_{(1.8,2.4)}=0.2\times T_{(1,2.4)}+0.8\times T_{(2,2.4)}=0.48 \tag{3-8}$$

从上述插值过程可以看出，双线性插值法对于输出图像中每个像素值的估计采用了其相邻 4 个像素的加权平均，故结果较最近邻插值法好，且不会出现“马赛克”现象。但由于每个像素都必须经过 6 次浮点运算才能获得较为准确的近似值，计算量较大，因而计算速度较慢。另外，当浮点数坐标像素相邻 4 个像素的值差别较大时，加权平均会导致图像边缘模糊化。

【贴士】 双线性插值的结果与插值的顺序无关。首先进行 y 方向的插值，然后进行 x 方向的插值，所得的结果是一样的。

3. 双三次插值法

双三次插值是一种更加复杂的插值方式，它能创造出比双线性插值更平滑的图像边缘。其基本原理是输出图像中每个像素的值都是由输入图像中相对应像素相邻 16 个像素值的加权平均得到的。

假设现有 5×5 大小的灰度图像，输出图像期望大小为 7×7，则所构建输出图像与输入图像之间的坐标对应关系如下：

$$\begin{cases} x_0 = \dfrac{5}{7}x \\ y_0 = \dfrac{5}{7}y \end{cases} \tag{3-9}$$

例如，输出图像中像素坐标(4,4)通过式(3-9)获得变换后的坐标(2.857,2.857)(红色点)，其相邻16个像素的坐标(蓝色点)，如图3-9所示。

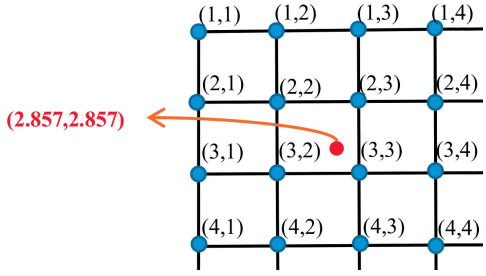


图3-9　双三次插值法原理示意图

双三次插值法使用BiCubic基函数来计算权值，其定义如下：

$$W(x) = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1, & |x| \leqslant 1 \\ a|x|^3 - 5a|x|^2 + 8a|x| - 4a, & 1 < |x| < 2 \\ 0, & \text{其他} \end{cases} \tag{3-10}$$

其中，参数 a 为常系数，通常取 -0.5；变量 x 为坐标(2.857,2.857)与相邻的16个像素之间的横向或纵向距离。以左上方第一个像素为例，坐标(2.857,2.857)与它的横向距离 $x=1.857$，则 $W(x)=(-0.5)|1.857|^3-5(-0.5)|1.857|^2+8(-0.5)|1.857|-4(-0.5)=-0.0089$；纵向距离 $y=1.857$，则 $W(y)=-0.0089$，因此第一个像素的权值为 $W(x)\cdot W(y)=0.00007921$。同理，求得其余15个像素对应的权值，再与各像素的值相乘求和，所得结果即为坐标(2.857,2.857)的像素近似值。

通过上述插值过程可以看出，双三次插值法对于输出图像的每个像素值的估计采用了其相邻16个像素的加权平均，计算复杂度上升，因而计算速度较慢。但该插值法由于考虑了相邻像素的影响度，使得插值后的图像清晰度因而得到了提高。

综上所述，最近邻插值法、双线性插值法和双三次插值法的性能特点各有优劣，如表3-1所示。在实际应用中，应根据应用需求进行选取，既要考虑时间方面的可行性，又要考虑插值后图像质量的要求，这样才能达到较为理想的结果。

表3-1　插值算法的性能比较

插值方法	性能特点
最近邻插值法	算法简单，运算速度较快。但图像质量损失较大，有明显的“马赛克”现象
双线性插值法	较最近邻插值法复杂，运算速度慢。基本克服了最近邻插值法的“马赛克”现象，但图像边缘在一定程度上较为模糊
双三次插值法	算法最为复杂，计算量最大；能够产生比双线性插值法更为平滑的边缘，计算精度很高，处理后的图像质量损失最小，效果是最佳的

3.2 仿射变换

仿射变换是一种二维坐标到二维坐标之间的线性变换，它保持了二维图形的“平直性”（即直线经变换后仍然是直线）和“平行性”（即直线之间的相对位置关系保持不变，平行线仍然是平行线，且直线上点的位置顺序不变）。在数字图像处理中，可应用仿射变换对二维图像进行平移、镜像、旋转、缩放、错切、转置等操作。

为了能够使用统一的矩阵线性变换形式来表示和实现这些常见的图像几何变换，需要引入一种新的坐标，即齐次坐标。

3.2.1 齐次坐标

在欧几里得空间即笛卡儿坐标系中，平行的直线不相交，但是在透视几何空间中，平行直线相交于无穷远点。为了方便在透视空间中处理图像，给坐标引入额外的一维 w，即齐次坐标，采用(x,y,w)来表示一个二维的齐次坐标，其对应的二维欧几里得坐标为$(x/w,y/w)$。

之所以称为齐次坐标，是因为对于任何 $w\neq0$ 的齐次坐标，在各个维度同除 w 后得到的是欧几里得空间中的一个点的坐标，如图 3-10 所示。另外，为了避免除法运算，常令 $w=1$，称为规范化的齐次坐标。

齐次坐标		笛卡儿坐标		
$(1,2,3)$	$\rightarrow$	$\left(\frac{1}{3},\frac{2}{3}\right)$		
$(2,4,6)$	$\rightarrow$	$\left(\frac{2}{6},\frac{4}{6}\right)$	$\rightarrow$	$\left(\frac{1}{3},\frac{2}{3}\right)$
$(4,8,12)$	$\rightarrow$	$\left(\frac{4}{12},\frac{8}{12}\right)$	$\rightarrow$	$\left(\frac{1}{3},\frac{2}{3}\right)$

图 3-10 齐次坐标与笛卡儿坐标的关系

3.2.2 图像几何变换的数学描述

齐次坐标表示的像素坐标是由 3 个元素组成的列向量，假设变换前输入图像中某像素的规范化齐次坐标矩阵为

$$\begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

变换后输出图像对应像素的规范化齐次坐标矩阵为

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

变换矩阵为

$$\boldsymbol{T}=\begin{bmatrix} a & b & p \\ c & d & q \\ l & m & s \end{bmatrix}$$

则图像几何变换公式可写为

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & p \\ c & d & q \\ l & m & s \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3-11}$$

这个 3×3 的变换矩阵 $\boldsymbol{T}$ 可以分成 4 个子矩阵。其中，$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$可使图像实现镜像、缩放、旋转和错切变换；$\begin{bmatrix} p \\ q \end{bmatrix}$可使图像实现平移变换；$[l \quad m]$可使图像实现透视变换，但当 $l=0, m=0$ 时无透视作用；$[s]$可使图像实现全比例变换，即

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ s \end{bmatrix} \tag{3-12}$$

将齐次坐标 $\begin{bmatrix} x_0 \\ y_0 \\ s \end{bmatrix}$规范化后得 $\begin{bmatrix} \frac{x_0}{s} \\ \frac{y_0}{s} \\ 1 \end{bmatrix}$。由此可见，当 $s>1$ 时，整幅图像按比例缩小；当 $0<s<1$ 时，整幅图像按比例放大；当 $s=1$ 时，整幅图像大小不变。

3.2.3　基本仿射变换

1. 平移变换

1）基本原理

平移变换是一种不产生形变而只移动的变换，图像上每个像素都移动相同的平移量。假设(x_0, y_0)是输入图像某像素坐标，水平平移量为 Δx，垂直平移量为 Δy，平移后坐标为(x, y)，如图 3-11 所示，则平移变换的坐标变换公式为

$$\begin{cases} x = x_0 + \Delta x \\ y = y_0 + \Delta y \end{cases} \tag{3-13}$$

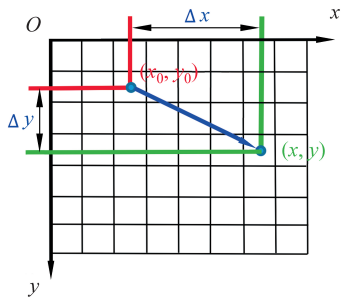


图 3-11　平移变换示意图

相应的齐次坐标矩阵表示为

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3-14}$$

因此，平移变换矩阵为

$$\begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \tag{3-15}$$

遍历输入图像中的每个像素，并按上述公式进行变换即可实现整幅图像的平移效果。

【贴士】　由于内部坐标系是连续坐标系，因此水平平移量 Δx 和垂直平移量 Δy 既可以取整数值也可以取实数值。

2）实现代码

平移变换的实现代码如下：

```
clear all
src=imread('rgb.jpg');
deltaX=-300;                                    %水平平移量
deltaY=200;                                     %垂直平移量
T=[1 0 deltaX;0 1 deltaY;0 0 1];                %平移变换矩阵
tform=affine2d(T');
result=imwarp(src,tform);                       %平移变换(内部坐标系)
subplot(1,2,1),imshow(src),title('输入图像');
subplot(1,2,2),imshow(result),title('内部坐标系下的平移效果');
```

【代码说明】

- affine2d()函数。该函数参数是变换矩阵的转置，这里需要对平移的变换矩阵做转置运算：

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

该函数的返回值是 affine2d 对象，具有变换矩阵和维度两个属性。代码如下：

```
tform=
  affine2d(带属性):
       T: [3x3 double]
       Dimensionality: 2
```

- imwarp()函数。

函数原型：

```
result=imwarp(src,tform);
```

其中，src 是输入图像，tform 是要应用的几何变换对应的 affine2d 对象。

运行以上代码，图 3-12(a)所示图像的平移变换效果如图 3-12(b)所示，但是可以观察到图像并没有产生平移的效果。

(a) 输入图像　　(b) 内部坐标系　　(c) 世界坐标系

图 3-12　图像的平移变换效果

此问题产生的原因在于图像的平移变换是相对的，是相对于参照物与周围环境形成的坐标空间而言的，因此在内部坐标系下，即使进行了仿射变换也观察不到平移效果。为此需将其放置于世界坐标系下观察。具体的编码中就是在调用 imwarp()函数时转换到世界坐标系，即将上述代码的第 8 行修改为

```
result=imwarp(src,tform,'OutputView',imref2d(size(src)));
```

在原有参数后加入由'OutputView'和 imref2d 空间参照对象组成的以“,”分隔的对组参数。这样在世界坐标系中就可以很直观地观察到其平移效果，如图 3-12(c)所示。

知识拓展

平移效果除了仿射变换外，还可以直接调用 imtranslate()函数实现，该函数原型为

```
B = imtranslate(A,translation);
```

其中，*A* 为输入图像矩阵；*B* 为输出图像矩阵；translation 为平移向量，由水平平移量和垂直平移量组成。

实现代码如下：

```
clear all
src=imread('rgb.jpg');
deltaX=-300;                                          %水平平移量
deltaY=200;                                           %垂直平移量
result=imtranslate(src,[deltaX deltaY]);              %平移变换
subplot(1,2,1),imshow(src),title('输入图像');
subplot(1,2,2),imshow(result),title('平移效果');
```

3）应用场景

【案例 3-1】重影滤镜。

（1）实现方法。对输入图像分别做左下、右上方向的平移变换，再对输入图像和两个平移结果图像进行加权平均求解。

（2）实现代码。重影滤镜的实现代码如下：

```
clear all
src=im2double(imread('flowers.jpg'));                 %读入输入图像
T1=[1 0 -20;0 1 20;0 0 1];                            %左下方向平移
tform1=affine2d(T1');
img1=imwarp(src,tform1,'OutputView',imref2d(size(src)));   %仅世界坐标系
T2=[1 0 20;0 1 -20;0 0 1];                            %右上方向平移
tform2=affine2d(T2');
img2=imwarp(src,tform2,'OutputView',imref2d(size(src)));   %仅世界坐标系
%求两个平移图像和输入图像的加权平均
result=0.2*img1+0.6*src+0.2*img2;
subplot(2,2,1),imshow(src),title('输入图像');
subplot(2,2,2),imshow(img1),title('左下方平移');
subplot(2,2,3),imshow(img2),title('右上方平移');
subplot(2,2,4),imshow(result),title('重影滤镜效果');
```

（3）实现效果。重影滤镜效果如图 3-13 所示。从图 3-13 的处理结果上看，平移操作导致输出图像边界处出现了不必要的边框，可以施以裁剪操作将其去除，如图 3-14 所示。

(a) 输入图像　(b) 左下方平移效果　(c) 右上方平移效果　(d) 加权平均效果

图 3-13　重影滤镜效果

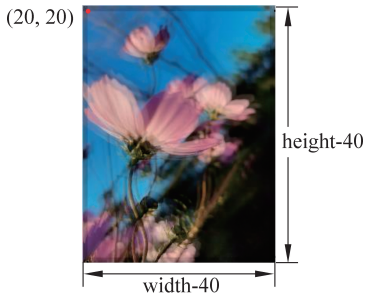


图 3-14　裁剪区域示意图

知识拓展

裁剪函数原型有以下两种语法形式。

形式 1：

```
B=imcrop(A,rect);
```

其中，*A* 是输入图像矩阵；*B* 是输出图像矩阵；rect 是由裁剪区域的左上角坐标 x、y、宽度及高度所组成的向量。

形式 2：

```
B=imcrop(A);
```

其中，*A* 是输入图像矩阵；*B* 是输出图像矩阵。

这两种形式的不同之处在于后者是交互式的裁剪方式，即裁剪区域是由用户使用鼠标交互选取的，而前者的裁剪区域是由 rect 向量指定的。

在案例 3-1 中，需要在重影处理之后添加以下代码以去除四周的边框：

```
width=size(result,2);
height=size(result,1);
result=imcrop(result,[20,20,width-40,height-40]);
```

小试身手

运用平移变换制作文字图像错位拼接的效果，具体要求详见习题 3 第 1 题。

【案例 3-2】故障风海报滤镜。

RGB 颜色分离故障又称颜色偏移故障，是故障艺术风格中比较常见的表达形式之一。例如，抖音短视频平台的图标就是 RGB 颜色分离故障艺术风格影响下的作品，给整体产品带来了潮流与年轻的气息。

（1）实现方法。本案例的基本实现思路是将一幅彩色图像的红、绿、蓝色分量分离，分别合成一幅仅有红色分量的彩色图像和一幅仅有蓝色分量的彩色图像，且对其中一幅做平移变换，最后再对二者做加法运算。

（2）实现代码。故障风海报滤镜的实现代码如下：

```
clear all
src=imread('test.jpg');
```

```
src_R=src(:,:,1);src_G=src(:,:,2);src_B=src(:,:,3);
%仅保留红色分量
img_R=cat(3,src_R,zeros(size(src,1),size(src,2)),zeros(size(src,1),size(src,2)));
%向右平移
T=[1 0 20;0 1 0;0 0 1];
tform=affine2d(T');
img_R=imwarp(img_R,tform,'OutputView',imref2d(size(src)));    %仅世界坐标系
%仅保留蓝色分量
img_B=cat(3,zeros(size(src,1),size(src,2)),zeros(size(src,1),size(src,2)),src_B);
%img_R与 img_B融合
result=imadd(img_R,img_B);
%裁边
width=size(result,2);
height=size(result,1);
result=imcrop(result,[20 0 width-20 height]);
subplot(2,2,1),imshow(src),title('输入图像');
subplot(2,2,2),imshow(img_R),title('仅红色分量');
subplot(2,2,3),imshow(img_B),title('仅蓝色分量');
subplot(2,2,4),imshow(result),title('故障风海报滤镜效果');
```

（3）实现效果。故障风海报滤镜效果如图 3-15 所示。

(a) 输入图像　(b) 仅红色分量图像　(c) 仅蓝色分量图像　(d) “融合+裁边”效果

图 3-15　故障风海报滤镜效果

2. 镜像变换

图像镜像变换分为水平镜像和垂直镜像。水平镜像是指以图像垂直中轴线为中心将图像左半部分和右半部分进行镜像对换；垂直镜像是指以图像水平中轴线为中心将图像上半部分和下半部分进行镜像对换。

1）水平镜像

（1）实现方法。水平镜像的基本原理如下：假设图像的高度为 height，宽度为 width，(x_0, y_0) 为输入图像某像素坐标，(x, y) 为经水平镜像变换后输出图像对应像素的坐标，如图 3-16 所示，则水平镜像变换的坐标变换公式为

$$\begin{cases} x = \text{width} - x_0 \\ y = y_0 \end{cases} \tag{3-16}$$

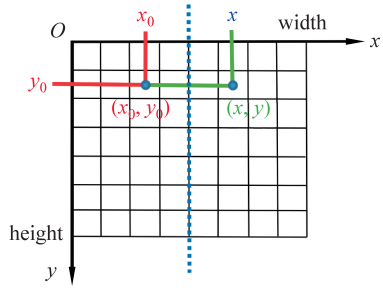


图 3-16　水平镜像变换原理示意图

相应的齐次坐标矩阵表示为

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & \text{width} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3-17}$$

因此，水平镜像变换矩阵为

$$\begin{bmatrix} -1 & 0 & \text{width} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3-18}$$

同样，遍历输入图像中的每个像素，并按照上述公式进行坐标变换即可实现整幅图像的水平镜像变换效果。

（2）实现代码。水平镜像的实现代码如下：

```
clear all
src=imread('rgb.jpg');
width=size(src,2);                                  %获取图像的宽度
T=[-1 0 width;0 1 0;0 0 1];                         %水平镜像变换矩阵
tform=affine2d(T');
%内部坐标系和世界坐标系均可
result=imwarp(src,tform,'OutputView',imref2d(size(src)));
subplot(1,2,1),imshow(src),title('输入图像');
subplot(1,2,2),imshow(result),title('水平镜像变换效果');
```

（3）实现效果。水平镜像效果如图 3-17 所示。

(a) 输入图像　　(b) 变换效果

图 3-17　水平镜像变换效果

知识拓展

除了仿射变换外，还可以直接调用内置函数 fliplr()或 flip()实现水平镜像变换效果，函数原型为

```
B = fliplr(A);
```

或

```
B = flip(A,2);
```

其中，A 为输入图像矩阵；B 为输出图像矩阵。

实现代码如下：

```
clear all
src=imread('rgb.jpg');
%水平镜像变换
result=fliplr(src);                                 %result=flip(src,2);
subplot(1,2,1),imshow(src),title('输入图像');
subplot(1,2,2),imshow(result),title('水平镜像变换效果');
```

2）垂直镜像

（1）实现方法。垂直镜像的基本原理如下：假设图像的高度为 height，宽度为 width，(x_0, y_0)为输入图像某像素坐标，(x, y)为经垂直镜像变换后输出图像对应像素的坐标，如图 3-18 所示，则垂直镜像变换的坐标变换公式为

$$\begin{cases} x = x_0 \\ y = \text{height} - y_0 \end{cases} \tag{3-19}$$

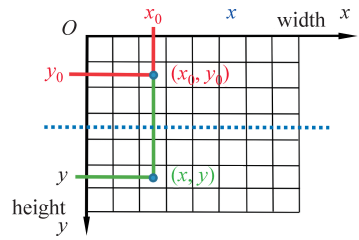


图 3-18　垂直镜像变换原理示意图

相应的齐次坐标矩阵表示为

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & \text{height} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3-20}$$

因此，垂直镜像变换矩阵为

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & \text{height} \\ 0 & 0 & 1 \end{bmatrix} \tag{3-21}$$

同样，遍历输入图像中的每个像素，并按照上述公式进行坐标变换即可实现整幅图像的垂直镜像变换效果。

（2）实现代码。垂直镜像的实现代码如下：

```
clear all
src=imread('rgb.jpg');
height=size(src,1);                          %获取图像的高度
T=[1 0 0;0 -1 height;0 0 1];                 %垂直镜像变换矩阵
tform=affine2d(T');
%内部坐标系和世界坐标系均可
result=imwarp(src,tform,'OutputView',imref2d(size(src)));
subplot(1,2,1),imshow(src),title('输入图像');
subplot(1,2,2),imshow(result),title('垂直镜像变换效果');
```

（3）实现效果。垂直镜像变换效果如图 3-19 所示。


(a) 输入图像　　(b) 变换效果

图 3-19　垂直镜像变换效果

知识拓展

垂直镜像效果除了仿射变换外，还可以直接调用内置函数 flipud()或 flip()实现，函数原型为

```
B = flipud(A);
```

或

```
B = flip(A,1);
```

其中，A 为输入图像矩阵；B 为输出图像矩阵。

实现代码如下：

```
clear all
src=imread('rgb.jpg');
垂直镜像变换
result=flipud(src);%result=flip(src,1);
subplot(1,2,1),imshow(src),title('输入图像');
subplot(1,2,2),imshow(result),title('垂直镜像变换效果');
```

小试身手

利用水平和垂直镜像的组合实现对角镜像，具体要求详见习题 3 第 2 题。

3）应用场景

【案例 3-3】蜡染壁纸制作。

（1）实现方法。

步骤 1：两次垂直镜像变换和上下拼接。首先对如图 3-20(a)所示的蜡染图案做垂直镜像变换得到变换图像，将蜡染图案和变换图像上下拼接成如图 3-20(b)所示的新图像；其次以这幅新图像为基准做第二次垂直镜像变换，同样将这幅新图像及其变换图像进行上下拼接，进而获得由 4 个蜡染图案组成如图 3-20(c)所示的新图案。

(a) 蜡染图案　　(b) 第一次垂直镜像和拼接　　(c) 第二次垂直镜像和拼接

图 3-20　垂直镜像变换与拼接效果

步骤 2：两次水平镜像变换和左右拼接。对如图 3-20(c)所示的图像做第一次水平镜像变换和左右拼接，在此基础上再做第二次水平镜像变换和左右拼接，最终获得由 16 个蜡染图案组成的 4×4 蜡染壁纸。

（2）实现代码。蜡染壁纸制作的实现代码如下：

```
clear all
src=imread('pattern.jpg');                                    %读入蜡染图案
img1=src;
%蜡染图案 img1 的垂直镜像变换和上下拼接
```

```
T1=[1 0 0;0 -1 size(img1,1);0 0 1];
tform1=affine2d(T1');
result1=imwarp(img1,tform1,'OutputView',imref2d(size(img1)));   %内部、世界坐标系均可
img2=[img1;result1];
%img2 的垂直镜像变换和上下拼接
T2=[1 0 0;0 -1 size(img2,1);0 0 1];
tform2=affine2d(T2');
result2=imwarp(img2,tform2,'OutputView',imref2d(size(img2)));   %内部、世界坐标系均可
img3=[img2;result2];
%img3 的水平镜像变换和左右拼接
T3=[-1 0 size(img3,2);0 1 0;0 0 1];
tform3=affine2d(T3');
result3=imwarp(img3,tform3,'OutputView',imref2d(size(img3)));   %内部、世界坐标系均可
img4=[img3 result3];
%img4 的水平镜像变换和左右拼接
T4=[-1 0 size(img4,2);0 1 0;0 0 1];
tform4=affine2d(T4');
result4=imwarp(img4,tform4,'OutputView',imref2d(size(img4)));   %内部、世界坐标系均可
result=[img4 result4];
%显示处理结果
subplot(2,3,1),imshow(src),title('蜡染图案');
subplot(2,3,2),imshow(img2),title('第一次垂直镜像和拼接');
subplot(2,3,3),imshow(img3),title('第二次垂直镜像和拼接');
subplot(2,3,4),imshow(img4),title('第一次水平镜像和拼接');
subplot(2,3,5),imshow(result),title('第二次水平镜像和拼接');
```

(3) 实现效果。蜡染壁纸效果如图 3-21 所示。

(a) 第一次水平镜像和拼接　　(b) 第二次水平镜像和拼接

图 3-21　水平镜像变换和拼接效果

小试身手

利用镜像变换制作万花筒特效，具体要求详见习题 3 第 3 题。

3. 旋转变换

1) 实现方法

图像的旋转变换是将图像绕任意点旋转指定角度以实现其位置的改变。

(1) 绕原点的逆时针旋转变换。假设(x_0,y_0)为输入图像某像素坐标，逆时针旋转角度α后其坐标为(x,y)，r表示像素坐标(x_0,y_0)与图像原点之间的距离。

由图 3-22 可以得到

$$\sin\beta=\frac{x_0}{r},\cos\beta=\frac{y_0}{r} \tag{3-22}$$

$$\sin(\alpha+\beta)=\frac{x}{r},\cos(\alpha+\beta)=\frac{y}{r} \tag{3-23}$$

经过推导得出，旋转变换的坐标变换公式为

$$\begin{cases}x=y_0\sin\alpha+x_0\cos\alpha\\ y=y_0\cos\alpha-x_0\sin\alpha\end{cases} \tag{3-24}$$

相应的齐次坐标矩阵表示为

$$\begin{bmatrix}x\\ y\\ 1\end{bmatrix}=\begin{bmatrix}\cos\alpha & \sin\alpha & 0\\ -\sin\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_0\\ y_0\\ 1\end{bmatrix} \tag{3-25}$$

因此，旋转变换矩阵为

$$\begin{bmatrix}\cos\alpha & \sin\alpha & 0\\ -\sin\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{bmatrix} \tag{3-26}$$

遍历输入图像中的每个像素，并按照上述公式进行旋转变换即可实现整幅图像的旋转变换效果。

（2）绕图像中心的逆时针旋转变换。绕图像中心的逆时针旋转本质上仍可看作是绕原点的逆时针旋转，只是原点的位置由(0,0)移动到了$(N/2,M/2)$，(x_0,y_0)相对于新原点的坐标为$(x_0-N/2,y_0-M/2)$，(x,y)相对于新原点的坐标为$(x-N/2,y-M/2)$，如图 3-23 所示。

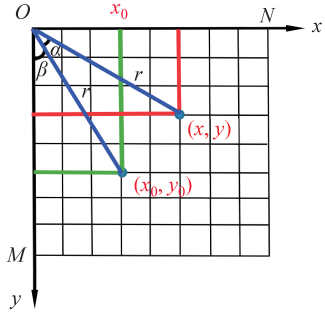


图 3-22　绕原点的逆时针旋转变换示意图

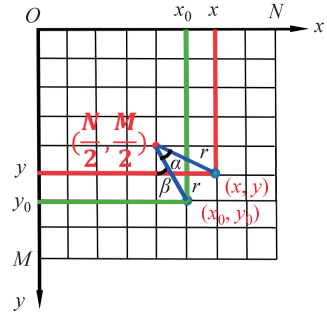


图 3-23　绕图像中心的逆时针旋转变换示意图

将$(x_0-N/2,y_0-M/2)$代入绕原点的逆时针旋转变换公式，可得

$$\begin{cases}x-\dfrac{N}{2}=\left(x_0-\dfrac{N}{2}\right)\cos\alpha+\left(y_0-\dfrac{M}{2}\right)\sin\alpha\\ y-\dfrac{M}{2}=-\left(x_0-\dfrac{N}{2}\right)\sin\alpha+\left(y_0-\dfrac{M}{2}\right)\cos\alpha\end{cases} \tag{3-27}$$

整理后得到

$$\begin{cases}x=\cos\alpha x_0+\sin\alpha y_0+\dfrac{N}{2}-\dfrac{N}{2}\cos\alpha-\dfrac{M}{2}\sin\alpha\\ y=-\sin\alpha x_0+\cos\alpha y_0+\dfrac{M}{2}-\dfrac{M}{2}\cos\alpha+\dfrac{N}{2}\sin\alpha\end{cases} \tag{3-28}$$

相应的齐次坐标矩阵表示为

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha & \frac{N}{2} - \frac{N}{2}\cos\alpha - \frac{M}{2}\sin\alpha \\ -\sin\alpha & \cos\alpha & \frac{M}{2} - \frac{M}{2}\cos\alpha + \frac{N}{2}\sin\alpha \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3-29}$$

因此，旋转变换矩阵为

$$\begin{bmatrix} \cos\alpha & \sin\alpha & \frac{N}{2} - \frac{N}{2}\cos\alpha - \frac{M}{2}\sin\alpha \\ -\sin\alpha & \cos\alpha & \frac{M}{2} - \frac{M}{2}\cos\alpha + \frac{N}{2}\sin\alpha \\ 0 & 0 & 1 \end{bmatrix} \tag{3-30}$$

2）实现代码

（1）绕原点的逆时针旋转变换。代码如下：

```
clear all
src=imread('rgb.jpg');
alpha=30*pi/180;                                                    %旋转角度(弧度)
%绕原点的逆时针旋转变换矩阵
T=[cos(alpha) sin(alpha) 0;-sin(alpha) cos(alpha) 0;0 0 1];
tform=affine2d(T');
result=imwarp(src,tform,'cubic','OutputView',imref2d(size(src)));   %仅世界坐标系
subplot(1,2,1),imshow(src),title('输入图像');
subplot(1,2,2),imshow(result),title('绕原点的逆时针旋转变换效果');
```

【代码说明】

由于旋转变换会产生“空洞”现象，因此需要在 imwarp()函数中第三个参数位置添加图像旋转所采用的插值方法，可以是'nearest'、'linear'或'cubic'，分别对应于最近邻插值法、双线性插值法和双三次插值法。

（2）绕图像中心的逆时针旋转变换。代码如下：

```
clear all
src=imread('rgb.jpg');
subplot(1,2,1),imshow(src),title('输入图像');
M=size(src,1);
N=size(src,2);
alpha=30*pi/180;                                                    %旋转角度(弧度)
T=[cos(alpha) sin(alpha) N/2-N/2*cos(alpha)-M/2*sin(alpha);-sin(alpha) cos(alpha)...
    M/2-M/2*cos(alpha)+N/2*sin(alpha);0 0 1];
tform=affine2d(T');
result=imwarp(src,tform,'cubic','OutputView',imref2d(size(src)));   %世界坐标系
subplot(1,2,2),imshow(result),title('绕图像中心的逆时针旋转变换效果');
```

3）实现效果

（1）图像绕原点的逆时针旋转变换效果如图 3-24 所示。

（2）图像绕图像中心的逆时针旋转变换效果如图 3-25 所示。

(a) 输入图像　　(b) 旋转效果

图 3-24　图像绕原点的逆时针旋转变换效果

(a) 输入图像　　(b) 旋转效果

图 3-25　图像绕图像中心的逆时针旋转变换效果

知识拓展

MATLAB 的图像处理工具箱中提供了内置函数 imrotate()实现绕图像中心的旋转效果。

（1）实现方法。imrotate()函数的语法格式如下：

```
B = imrotate(A,angle,method,bbox);
```

其中，A 为输入图像矩阵，B 为输出图像矩阵，angle 为旋转角度（大于 0 表示逆时针，小于 0 则表示顺时针），method 为采用的插值方法（可以是'nearest'、'bilinear'、'bicubic'分别对应于最近邻插值法、双线性插值法和双三次插值法），bbox 为显示方式（可以是'loose'、'crop'）。

（2）实现代码：

```
clear all
src=imread('rgb.jpg');
result1=imrotate(src,30,'nearest','loose');
result2=imrotate(src,30,'nearest','crop');
subplot(1,3,1),imshow(src),title('输入图像');
subplot(1,3,2),imshow(result1),title(''loose'旋转效果');
subplot(1,3,3),imshow(result2),title(''crop'旋转效果');
```

（3）实现效果。图 3-26(a)所示图像的'loose'旋转效果如图 3-26(b)所示，'crop'旋转效果如图 3-26(c)所示。

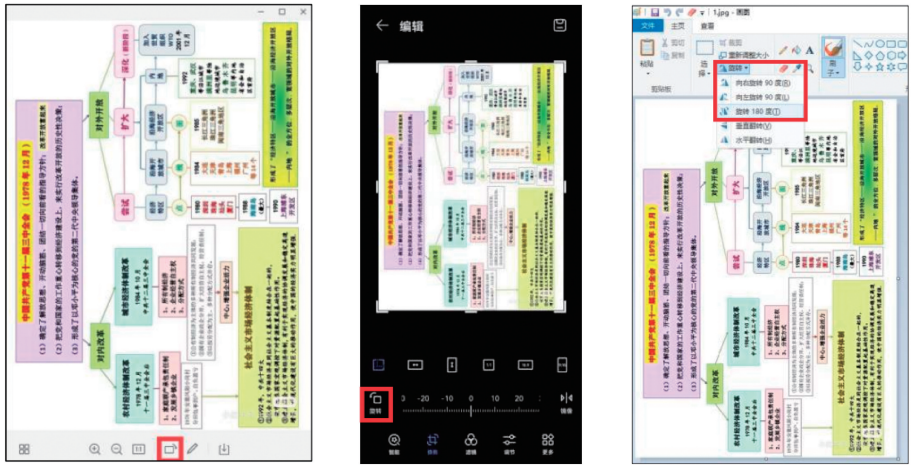
(a) 输入图像　(b) 'loose'旋转效果　(c) 'crop'旋转效果

图 3-26　函数 imrotate()的图像旋转效果

4) 应用场景

【案例 3-4】图片编辑器之旋转功能模拟。

在日常生活中经常会遇到图片角度不正常，无法满足实际使用需求，这时就需要借助一些图片编辑工具中的旋转功能来进行角度调整。图 3-27(a)为微信工具、图 3-27(b)为华为手机图库工具、图 3-27(c)为绘图软件中相应的旋转功能。在本案例中，模拟常见的“向左旋转 90°”和“向右旋转 90°”两大功能。



(a) 微信图片编辑工具　(b) 华为手机图库工具　(c) 绘图软件

图 3-27　各类工具中的旋转功能

(1) 实现方法。设计基于 MATLAB 的 GUI 图形化用户界面，实现用户交互式“向左旋转 90°”“向右旋转 90°”的图片旋转功能以及保存图片功能。具体操作如下。

步骤 1：在 MATLAB 开发环境下，在“主页”选项卡中选中“新建”|“图形用户界面”选项，在弹出的“GUIDE 快速入门”窗口中选中“新建 GUI”选项卡左侧 GUIDE templates 中的 Blank GUI(Default)，并单击“浏览”按钮，在弹出的对话框中选取存储位置，将 GUI 窗口名称命名为 main.fig 后选中此项，最后单击“确定”按钮，如图 3-28(a)所示。

步骤 2：向 GUI 窗口中添加两个轴控件 axes1、axes2 和 3 个按钮控件 pushbutton1、pushbutton2、pushbutton3，并设置 pushbutton1、pushbutton2、pushbutton3 的 Tag 属性分别为 pushbuttonCCWRotate、pushbuttonCWRotate、pushbuttonSave，FontSize 属性均为 12，String 属性分别为“向左旋转 90°”“向右旋转 90°”“保存图片”，如图 3-28(b)所示。