第Ⅱ部分　图像处理实践

在本部分中，我们将主要聚焦于数字图像处理技术，并结合实践操作来加深对相关理论的理解。具体来说，这一部分将涵盖基础操作、几何变换、图像滤波、边缘检测以及特征提取与匹配五大主题。通过这一系列的学习，读者能够解决图像处理领域的一些常见问题，并实现图像特征的有效提取和分析。

第 3 章　图像的基本操作

3.1　概述

本章主要介绍图像的基本知识，包括像素、图像分类、颜色空间等概念，同时介绍计算机读写图像的基本原理，在计算机中，图像是以矩阵的形式存储，大小由图像的宽度和高度决定，所以对图像的读写实质上是对矩阵的处理。引入开源的计算机视觉库 OpenCV，它是一个功能强大且易于使用的图像处理和分析的软件包，提供了多种编程语言和平台的支持，通过对 OpenCV 提供的算法工具学习，使用 OpenCV 提供的函数来实现对图像的读取、显示和保存。这些操作既是图像处理的基础，也是后续学习更高级的图像分析和应用的前提，因此，本章对这些基本操作的系统学习和实践对后续内容的学习意义重大。

本章末尾给出的实践习题有一定挑战性，学有余力的读者可以挑战一下自己，更深刻地理解这部分的知识。

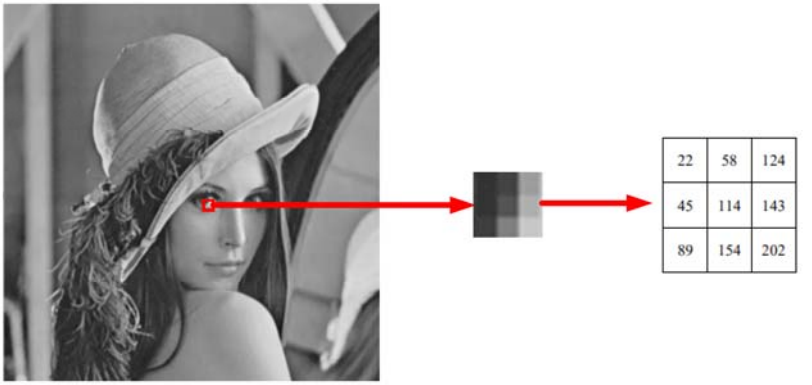
3.2　图像及基本操作

我们将从两个层面来介绍图像理论基础及其基本操作。

3.2.1 图像

图像作为一种表达信息的媒体，能够包含非常丰富的信息，具有直观性强、易于理解的特点，能够通过色彩、构图等元素传达复杂的含义。科学研究表明，人脑从周围环境获取的信息中，约有 95% 来自视觉。

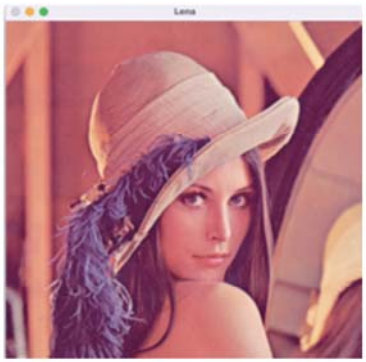
常见的数字图像类型包括二值图像、灰度图像、彩色图像、多光谱图像、深度图像和矢量图像等，我们以常见的灰度图像和彩色图像为例进行重点介绍。在计算机中，灰度图像是用矩阵来表示的，一个矩阵元素代表灰度图像上的一个点，也即灰度图像的像素。一个像素值代表了对应位置的亮度，这个值的范围通常为 [0,255](无符号 8 位整数)，像素值越大，对应的灰度图像位置也就越亮。图 3-1 中的灰度图像示意图可以帮助你加深理解。



(a) 原图　　(b) 图像局部　　(c) 局部图像的灰度值

图 3-1　灰度图像示意图

较常见的 RGB 色彩模式利用一个三维数组来表示一幅图像，3 个通道分别代表 R、G、B 三个分量，R 表示红色，G 表示绿色，B 表示蓝色，通过这三种基本颜色就可以合成任意颜色，每个通道的值在 0 到 255 之间，颜色的强度 (即每个通道的值) 表示了人眼对颜色的感知程度，每个像素的颜色由三个通道的值组合而成。需要特别指出的是，OpenCV 是按 BGR 的顺序读取图像的，而不是 RGB 顺序，感兴趣的同学可以进一步探究。同样的，图 3-2 中的 RGB 图像示意图可以帮助你加深理解。

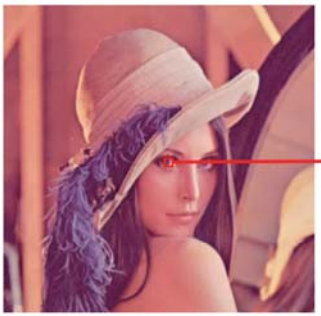


(a) 原图

(b) 彩色图像的 R 分量

(c) 彩色图像的 G 分量

(d) 彩色图像的 B 分量



(e) 原图

(f) 图像局部

R:111 G:44 B:84	R:122 G:48 B:94	R:147 G:89 B:132
R:116 G:49 B:90	R:125 G:41 B:102	R:153 G:25 B:87
R:134 G:38 B:87	R:116 G:50 B:62	R:90 G:32 B:45

(g) 局部图像的 RGB 值

图 3-2 RGB 图像

图像数值是指图像中每个像素的灰度或颜色值，可以用矩阵来表示。在计算机内存中，这个矩阵是按照一定的顺序存储的，通常是从左到右，从上到下，也就是说，第一个元素是图像左上角的像素值，最后一个元素是图像右下角的像素值。这种存储方式也意味着图像的左上角是坐标系的原点，x 轴和 y 轴分别沿着水平和垂直方向延伸。当然，也有一些图像格式采用从左到右，从下到上的存储顺序，这时候图像的左下角就是坐标系的原点，x 轴和 y 轴的方向不变。对于彩色图像，每个矩阵元素有 R(红色)、G(绿色) 和 B(蓝色) 三个分量 (三个通道)。矩阵元素在计算机中的存储顺序与灰度图像相同，同时每个元素的分量按照 BGR(以 OpenCV 为例) 的顺序进行存放。图 3-3 和图 3-4 分别表示了灰度图像和 RGB 图像在计算机中的存储示意图。

视频是由一系列的图像组成的，每个图像称为一帧，如常见的 30 帧视频就是表示在一秒钟内展现 30 张图片，视频的播放是通过快速切换不同的帧来实现的，给人一种连续的视觉效果。当然，这样的理解并不完全准确，因为我们平时接触到的视频都会进行压缩和解码以及音频信息同步等一系列操作，并不能单纯理解为图片的叠加，但对初学者理解本书并无影响，故在此不作展开。

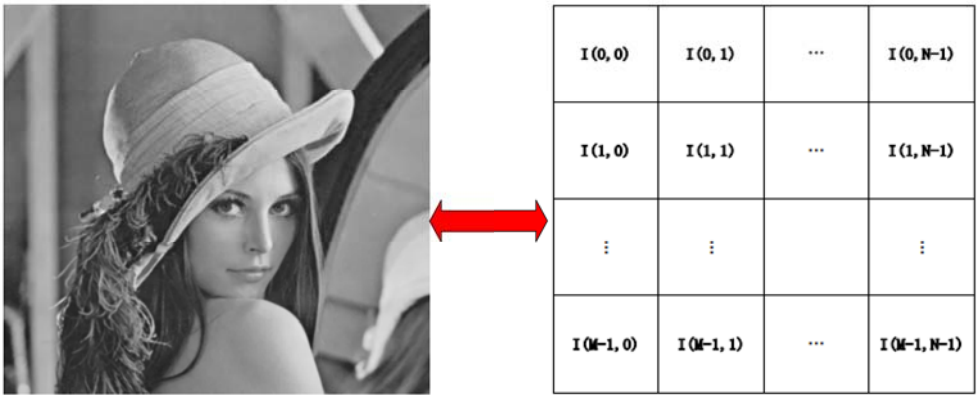


图 3-3　灰度图像存储

{B(0, 0)	G(0, 0)	R(0, 0)}	{B(0, 1)	G(0, 1)	R(0, 1)}	…	{B(0, N-1)	G(0, N-1)	R(0, N-1)}
{B(1, 0)	G(1, 0)	R(1, 0)}	{B(1, 1)	G(1, 1)	R(1, 1)}	…	{B(1, N-1)	G(1, N-1)	R(1, N-1)}
⋮	⋮	⋮	⋮	⋮	⋮		⋮	⋮	⋮
{B(M-1, 0)	G(M-1, 0)	R(M-1, 0)}	{B(M-1, 1)	G(M-1, 1)	R(M-1, 1)}	…	{B(M-1, N-1)	G(M-1, N-1)	R(M-1, N-1)}

图 3-4　RGB 图像存储

3.2.2　OpenCV 中的图像基本操作函数

OpenCV 本身提供了 cv2.imread(), cv2.imwrite() 和 cv2.imshow() 来处理图像文件的读取、写入和显示。我们可以在自己的代码中利用 OpenCV 提供的这些库函数来实现所需的功能。

(1) cv2.imread 函数是 OpenCV 库中的一个函数，用于从文件中读取图像。这里的 cv2 指的是 OpenCV，由一系列 C 函数和少量 C++ 构成，在安装这个库的时候，名称为 OpenCV Python，但在导入库的时候要写作 import cv2。

```
retval = cv2.imread(filename，flag)
```

- retval：读取的图像数据。
- filename：读取文件的路径。
- flag：可选参数有很多，表示读取图像的模式。例如，-1 表示以 RGB 模式读取，0 表示以灰度模式读取，1(默认) 表示以 BGR 模式读取。函数返回一个 numpy 数组，表示图像的像素值。
- flag 扩展
  - cv2.IMREAD_REDUCED_GRAYSCALE_2：将原图转为单通道灰度图，并且尺寸缩小为原始的 1/2。
  - cv2.IMREAD_REDUCED_COLOR_2：将原图转为三通道的 BGR 图像，并且尺寸缩小为原始的 1/2。
  - cv2.IMREAD_REDUCED_COLOR_4：将原图转为三通道的 BGR 图像，并且尺寸缩小为原始的 1/4。
  - cv2.IMREAD_REDUCED_COLOR_8：将原图转为三通道的 BGR 图像，并且尺寸缩小为原始的 1/8。

(2) cv2.imwrite 函数是 OpenCV 库中的一个函数，用于将图像保存到文件中。示例如下。

```
cv2.imwrite(filename,image)
```

- filename：参数是保存文件的路径。
- image：参数是一个 numpy 数组，表示图像的像素值。函数返回一个布尔值，表示保存是否成功。

(3) cv2.imshow 函数是 OpenCV 库中的一个函数，用于在窗口中显示图像。示例见 3.3.2 节的代码实现部分。

```
cv2.imshow(window_name,image)
```

- window_name：参数是窗口的名称。
- image：参数是一个 numpy 数组，表示图像的像素值。函数没有返回值。要显示图像，还需要使用 cv2.waitKey 函数来等待用户按键，并使用 cv2.destroyAllWindows 函数来关闭窗口。

(4) cv2.waitKey 函数是 OpenCV 库中的一个函数，用于等待用户按键。示例见 3.3.2 节的代码实现部分。

```
cv2.waitKey(delay)
```

delay 参数是等待时间，以毫秒为单位。如果参数为正数，函数会在指定的时间内等待任意键的输入。如果参数为 0 或负数，函数会无限期地等待键盘输入。函数返回按下的键的编码，如果没有按键，则返回 -1。

(5) cv2.VideoCapture 是 OpenCV 中的一个类，用于从视频文件、图像序列或摄像头捕获视频。可以使用 cv2.VideoCapture 的构造函数来指定要打开的视频源，例如：

- cv2.VideoCapture(0)：表示打开笔记本的内置摄像头。
- cv2.VideoCapture("../test.avi")：表示打开指定路径下的视频文件。

可以使用 cv2.VideoCapture 的 read 方法来按帧读取视频，返回一个布尔值和一个图像对象。

```
retval, image = cv2.VideoCapture.read()
```

- retval：布尔值，获取到视频帧为 True，否则为 False。
- image：获取到的视频帧。如果未读取到视频帧则为空。

(6) cv2.VideoWriter 类用于创建视频文件。示例见 3.3.4 节的代码实现部分，与读视频不同的是，需要在创建视频时设置一系列参数，包括视频文件名、编码解码格式、帧率、视频宽度和高度等。

```
cv2.VideoWriter(filename, fourcc, fps, framesize[, iscolor])
```

- filename ：要创建的视频文件名。

- fourcc：以 4 个字符表示视频压缩的编解码器。
- fps：视频帧率。
- framesize：视频的尺寸。
- iscolor：如果非 0 编码器将按彩色帧进行编码，否则按灰度帧进行编码。

3.3 图像基本操作示例

示例如下。

3.3.1 实验准备

本章实践所使用的硬件和软件环境请参照第 I 部分实践环境部分进行配置。

3.3.2 图像读写实例

本实验所需文件：image_readwrite.py

本实验依赖库：OpenCV-Python (即 cv2)

3.3.2.1 代码实现

代码实现如下。

```
#!/usr/bin/env python3
# encoding:utf-8
import cv2 as cv                    # 导入 opencv 库以调用图像处理的函数
def main():
    # 读取图像
    im = cv.imread("lena.jpg")
    # 读取图像，同时转换为灰度图
    im_gray = cv.imread("lena.jpg", cv.IMREAD_GRAYSCALE)
    # 读取图像，同时将图像大小缩小为原始大小的 1/2
    im_small = cv.imread("lena.jpg", cv.IMREAD_REDUCED_COLOR_2)
```

```
    # 将上面缩小的图像写入文件
    cv.imwrite("lena_small.jpg", im_small)
    # 显示图像
    cv.imshow("Lena", im)
    cv.imshow("Lena_Gray", im_gray)
    cv.imshow("Lena_small", im_small)
    cv.waitKey()
    cv.destroyAllWindows()
if __name__ == '__main__':
    main()
```

3.3.2.2 运行示例

我们可以按照下面的步骤进行实践。

(1) 首先按照第 I 部分要求进行硬件和软件环境配置，如果环境已经配置，本步可以跳过。

(2) 通过 cd 指令进入到存放有 image_readwrite.py 的文件目录下 (假定文件按照第 I 部分的路径组织)，该文件目录处于 Ubuntu 系统的桌面中的 examples 母文件夹中的子文件夹 01，如图 3-5 所示。实际操作中读者可根据具体文件所在位置进入对应的路径下。

图 3-5 进入指定路径

(3) 使用 python3 命令运行 image_readwrite.py 文件，如图 3-6 所示。

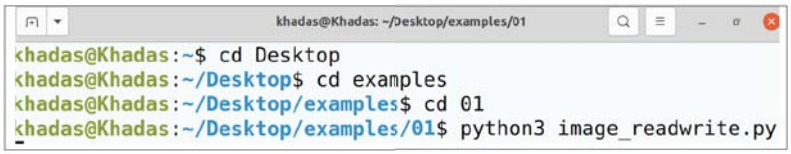


图 3-6 运行相应文件

(4) 实验结果。

在 image_readwrite.py 代码中，通过 OpenCV 提供的库函数 cv2.imread("lena.jpg") 来读取名为“lena.jpg”的图片，并通过库函数 cv2.imshow("Lena", im) 来进行显示，结果如图 3-7 图 (a) 所示。通过 3.2.2 节对 cv2.imread 函数的介绍，可以通过更改函数中的 flag 参数来实现对图片的不同读取，例如 cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE) 可以将原图片转换为灰度图，结果如图 3-7(b) 所示；cv2.imread("lena.jpg", cv2.IMREAD_REDUCED_COLOR_2) 可以读入图像，同时将图像大小缩小为原始大小的 1/2，读写进行结果如图 3-7(c) 所示。在介绍 cv2.imread 库函数时，我们介绍了 flag 参数的常用扩展，感兴趣的读者可以自行体验图像在不同参数下的显示结果。

(a) RGB 图　　(b) 灰度图　　(c) 1/2 RGB 图

图 3-7　图像读写运行结果

3.3.3　视频读取实例

本实验所需文件：video_read.py

本实验依赖库：OpenCV-Python (即 cv2) 和 Sys

3.3.3.1 代码实现

代码实现如下。

```
#!/usr/bin/env python3
# encoding:utf-8
import sys
import cv2 as cv                                    # 导入 opencv 库以调用图像处理的函数
def main():
    # 打开第一个摄像头
    cap = cv.VideoCapture(0)
    # 打开视频文件
    #cap = cv.VideoCapture("vtest.avi")
    # 检查是否打开成功
    if cap.isOpened() == False:
        print('Error opening the video source.')
        sys.exit()
    while True:
        # 读取一帧视频，存放到 im
        ret, im = cap.read()
        if not ret:
            print('No image read.')
            break
        # 显示视频帧
        cv.imshow('Live', im)
        # 等待 30 毫秒，如果有按键则退出循环
        if cv.waitKey(30) >= 0:
            break
    # 销毁窗口
    cv.destroyAllWindows()
    # 释放 cap
    cap.release()
if __name__ == '__main__':
        main()
        print('--(!)Error opening video capture')
        exit(0)
```

3.3.3.2 运行示例

在 3.3.2.1 节的基础上，可以直接执行下面的操作步骤。如果设备断电关机，则需要参考 3.3.2 节中实验运行示例部分的操作。

我们可以按照下面的步骤进行实践。

(1) 首先按照第 I 部分要求进行硬件和软件环境配置，如果环境已经配置，本步可以跳过。

(2) 通过 cd 指令进入到存放有 video_read.py，的文件目录下 (假定文件按照第 I 部分的路径组织)，该文件目录处于 Ubuntu 系统的桌面中的 examples 母文件夹中的子文件夹 01，如图 3-8 所示。实际操作中，可根据具体文件所在位置进入对应的路径下)。

图 3-8 进入指定路径

(3) 使用 python3 命令运行 video_read.py 文件，如图 3-9 所示。

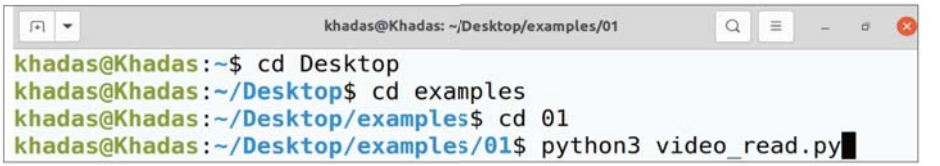


图 3-9 运行相应文件

(4) 实验结果。

我们通过 OpenCV 的库函数 cv2.VideoCapture(0) 来调用设备机械臂上的摄像头，通过 ret, im = cap.read() 来获取摄像头读取的数据，通过 OpenCV 的延时函数 cv2.waitKey(30) 来使每一帧图片显示 30 毫秒，我们就实现了对于视频的读取和显示。结果如图 3-10 所示，当图像显示窗口激活时，按任意键退出程序。

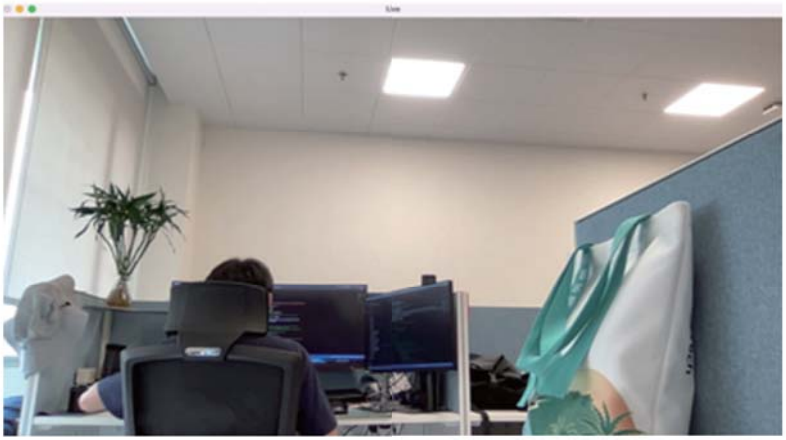


图 3-10　视频读取示例运行结果

3.3.4　视频文件创建实例

本实验所需文件：video_write.py

本实验依赖库：OpenCX-Python(即 cv2)、Sys 和 NumPy

3.3.4.1　代码实现

代码实现如下。

```
#!/usr/bin/env python3
# encoding:utf-8
import sys
import numpy as np
import cv2 as cv
def main():
    # 设置视频的宽度和高度
    frame_size = (320, 240)
    # 设置帧率
    fps = 25
```

```
    # 视频编解码格式
    fourcc = cv.VideoWriter_fourcc('M','J','P','G')
    # 创建 writer
    writer = cv.VideoWriter("myvideo.avi", fourcc, fps, frame_size)
    # 检查是否创建成功
    if writer.isOpened() == False:
        print("Error creating video writer.")
        sys.exit()
    for i in range(0, 100):
        # 设置视频帧画面
        im = np.zeros((frame_size[1], frame_size[0], 3), dtype=np.uint8)
        # 将数字绘制到画面上
        cv.putText(im,str(i),(int(frame_size[0]/3),int(frame_size[1]*2/3)), cv.FONT_
HERSHEY_SIMPLEX, 3.0, (255, 255, 255), 3)
        # 保存视频帧到文件 "myvideo.avi"
        writer.write(im)
    # 释放 writer
    writer.release()
if __name__ == '__main__':
    main()
```

3.3.4.2 运行示例

在前面实验的基础上，我们可以直接进行下面的操作步骤。如果设备断电关机，则需要参考 3.3.2 节中实验运行示例部分进行操作。

我们可以按照下面的步骤进行实践。

(1) 首先按照第 I 部分要求进行硬件和软件环境配置，如果环境已经配置，本步可以跳过。

(2) 通过 cd 指令进入到存放有 video_write.py，的文件目录下 (假定文件按照第 I 部分的路径组织)，该文件目录处于 Ubuntu 系统的桌面中的 examples 母文件夹中的子文件夹 01, 如图 3-11 所示。实际操作中读者可根据具体文件所在位置进入对应的路径下)。

图 3-11　进入指定路径

(3) 使用 python3 命令运行 video_write.py 文件，如图 3-12 所示。

图 3-12　运行相应文件

(4) 实验结果。

我们通过 frame_size = (320, 240) 来设置视频的宽度和高度，通过 fps = 25 来设置帧率，通过 fourcc = cv.VideoWriter_fourcc('M','J','P','G') 来设置视频编解码格式，writer = cv.VideoWriter("myvideo.avi", fourcc, fps, frame_size) 的第一个参数为创建的视频名称 (可以修改)，通过 for i in range(0, 100) 来定义视频的总帧数 (也可以理解为时长)，视频画面为 im=np.zeros((frame_size[1],frame_size[0],3),dtype=np.uint8)，cv.putText(im,str(i),(int(frame_size[0]/3),int(frame_size[1]*2/3)) 是将数字绘制到画面上，最终创建了一个名为 myvideo.avi 的文件。结果如图 3-13 所示。

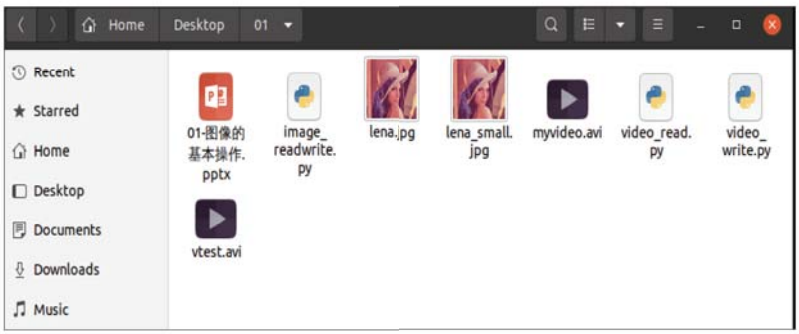


图 3-13　视频文件创建示例

3.4 小结

在本章中，我们从最基础的图像处理入手，开始介绍 OpenCV 库提供的图像操作功能。首先学习了读取和保存图像文件，这是图像处理的基础，任何处理都必须建立在读取图像的基础上。然后学习了显示图像的功能，这能够帮助我们进行调试和观察图像内容。

除了静态图像外，还介绍了如何读取视频文件以及调用计算机的摄像头实时获取图像流。这为处理动态图像如视频提供了支持。所有这些基本操作构成了我们利用 OpenCV 库进行图像处理的基础框架。通过掌握这些内容，我们可以利用 OpenCV 丰富的函数库和数据结构，进行更复杂的图像检测、识别和理解等任务。

这些内容为我们构建更复杂的图像处理算法和应用奠定了基础，理解和掌握这些内容有助于你更深入地探索图像处理的各个方面，并应用于更广泛的图像处理任务和应用领域。

3.5 实践习题

(1) 用 OpenCV 提供的函数实现图片的不同读入方法，并理解其实现方式。可参考 3.2.2 节中的“flag 扩展”部分。

(2) 体验视频读取参数对显示结果的影响。

(3) 请用 OpenCV 读取摄像头的视频流并显示，同时保存一帧视频帧和视频流。

第 4 章　图像的几何变换

4.1　概述

图像的几何变换是一种图像处理技术，它的目的是改变图像中物体的位置、形状和大小，以适应不同的应用场景。图像的几何变换通过将原始图像中的每个像素点按照一定的数学规则映射到新的坐标位置，从而生成一幅新的图像。在这个过程中，图像中的像素值不会发生变化，只是它们所处的空间位置发生了变化。常见的几何变换有很多种，例如平移、旋转、缩放、翻转、错切等。在本章，仍然是通过 OpenCV 提供的函数来实现一系列的图像几何变换操作。

在本章末尾给出有一定挑战性的实践习题，希望学有余力的读者可以挑战一下自己，可以使读者更深刻地理解这一部分的知识。

4.2　图像几何变换基础

下面要介绍相关理论基础。

4.2.1　几何变换

图像的几何变换是指将一幅图像中的坐标映射到另外一幅图像中的新坐标位置，它

不改变图像的像素值，只是改变像素所在的几何位置，使原始图像按照需要产生位置、形状和大小的变化。常见的几何变换包括平移、旋转、缩放、翻转、错切等。平移是将图像沿着水平或垂直方向移动一定的距离；旋转是将图像绕着某个点或某个轴旋转一定的角度；缩放是将图像放大或缩小一定的比例；翻转是将图像沿着水平或垂直方向反转；错切是将图像沿着水平或垂直方向拉伸或压缩。

4.2.2 几何变换原理

图像几何变换是建立在矩阵运算基础上的，通过矩阵运算可以很快找到对应关系。

4.2.2.1 平移

将图像沿着水平或垂直方向移动一定的距离，不改变图像的大小和形状。假设变换前的坐标为 (x, y)，变换后的坐标为 (u, v)，则有 $u = x + \Delta x$, $v = y + \Delta y$，写成矩阵形式为：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4-1}$$

4.2.2.2 旋转

图像的几何变换中的旋转是指将图像绕图像中心顺时针或逆时针旋转一定角度。设初始坐标为 (x, y) 的点经过旋转后坐标变为 (u, v)，其中 $u = x\cos\theta - y\sin\theta$，$v = x\sin\theta + y\cos\theta$。写成矩阵形式为：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4-2}$$

4.2.2.3 缩放

缩放变换是几何变换中的一种，它是一种将图像沿着坐标轴方向进行放大或缩小的

变换，即将图形中每个点的坐标分别乘以一个比例因子，从而得到新的图形。设初始坐标为 (x, y) 的点经过缩放后坐标变为 (u, v)，其中 $u = \sigma_x * x$，$v = \sigma_y * y$。写成矩阵形式为：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \sigma_x & 0 & 0 \\ 0 & \sigma_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4-3}$$

4.2.2.4 翻转

在几何变换中，翻转是一种基本的变换之一，它可以分为水平翻转和垂直翻转两种类型。水平翻转是指将物体从左向右或从右向左旋转 180 度，垂直翻转是指将物体从上向下或从下向上旋转 180 度。以水平翻转为例，设初始坐标为 (x, y) 的点经过水平翻转后的坐标变为 (u, v)，则有 $u = -x$，$v = y$，写成矩阵形式为：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4-4}$$

4.2.2.5 错切

错切是一种几何变换，它可以将一个二维图像沿着某个方向平移一定的距离，同时使该图像在与平移方向垂直的方向上产生拉伸或压缩。以横向错切为例，设初始坐标为 (x, y) 的点经过水平翻转后的坐标变为 (u, v)，则有 $u = x + s_x * y$，$v = y$，写成矩阵形式为：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & s_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4-5}$$

4.2.3 插值原理

在图像进行各种几何变换时，目标图像中每个像素颜色值需要从原始图像中找到相

应的像素颜色值进行填充，即对于目标图像的像素要找到其在原始图像的像素进行填充。但在实际矩阵变换过程中，映射过来的坐标值不一定是整数，那么此时使用原始图像像素填充，需要有不同的处理算法。

在图像几何变换的过程中，常用的插值方法有最近邻插值、双线性内插值和三次卷积法插值。

4.2.3.1 最近邻插值

这是一种最为简单的插值方法，在图像中最小的单位就是单个像素，但是在旋转缩放的过程中如果出现了小数，那么就对这个浮点坐标进行简单的取整，得到一个整数型坐标，这个整数型坐标对应的像素值就是目标像素的像素值。取整的方式是指取浮点坐标最近邻的左上角的整数点。

4.2.3.2 双线性内插值

对于一个目的像素，设置坐标通过反向变换得到的实数标为 $(i+u, j+v)$，其中 i、j 均为非负整数，u、v 为 [0,1) 区间的浮点数，则这个像素的值 $f(i+u, j+v)$ 可由原图像中坐标为 (i,j)、$(i+1,j)$、$(i,j+1)$、$(i+1,j+1)$ 所对应的周围四个像素的值决定。图 4-1 的双线性内插值原理图可以帮助你更好地理解。

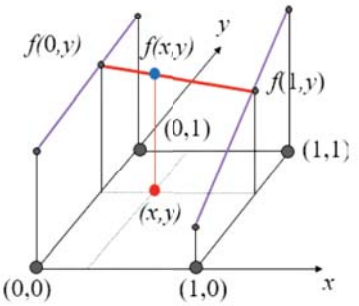


图 4-1 双线性内插值原理图

公式表示如下：

$$f(0, y)=f(0, 0)+y\,[f(0, 1)-f(0, 0)] \tag{4-6}$$

$$f(1, y)=f(1, 0)+y\,[f(1, 1)-f(1, 0)] \tag{4-7}$$

$$f(x, y)=f(0, y)+x\,[f(0, y)-f(0, y)] \tag{4-8}$$

4.2.3.3 三次卷积插值

三次卷积插值可以用于二维图像中的插值。在二维情况下，需要在每个方向上取 3 个离散数据点进行卷积运算，即可得到该点的插值。

具体而言，假设需要在二维平面上对一个位置为 (x, y) 的离散数据进行插值，那么可以从该位置的周围取出 3×3 的像素块，共取出 9 个离散数据点。然后，对于该点进行水平方向和垂直方向上的三次卷积运算，即可计算出该点的插值。表 4-1 显示近邻插值法、双线性插值法和三次卷积法的优劣对比。

表 4-1 几种插值法优劣对比

插值方法	优点	缺点
最近邻插值法	算法简单，计算速度快	插值效果差，容易出现锯齿状的伪像和失真
双线性内插值法	比近邻插值法更精确，能够抑制锯齿状伪像和失真	计算复杂度较高，需要处理边界插值问题
三次卷积插值法	插值效果较好，能够较好地处理锐利边缘和细节	计算复杂度相对较高，需要处理边界插值问题

注意，不同的插值方法在不同的应用场景下都有其优缺点。因此，在选择插值方法时需要综合考虑应用场景和需要达到的效果。

4.2.4 OpenCV 中的几何变换函数

下面我们提供了 OpenCV 中的常用的几何变换函数。

(1) cv2.warpAffine 是 OpenCV 库中的一个函数，用于对图像进行仿射变换。

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

- src：表示输入图像。
- M：表示变换矩阵。可以通过改变 M 的值来实现我们所需的一系列几何变换。
- dsize：表示输出图像的大小，二元元组 (width, height)。
- dst：表示变换操作的输出图像，可选项。
- flags：表示插值方法，整型 (int)，可选项。
- borderMode：表示边界像素方法，整型 (int)，可选项，默认值为 cv.BORDER_REFLECT。
- borderValue ：表示边界填充值，可选项，默认值为 0(黑色填充)。

(2) cv2.resize 是 OpenCV 库中的一个函数，用于调整图像的大小。

cv2.resize(src, dst, dsize, fx=0, fy=0, interpolation=INTER_LINEAR)

- src：输入，原图，即待改变大小的图像。
- dst：输出，改变后的图像。
- dsize：输出图像的大小。
- fx：width 方向的缩放比例。
- fy：height 方向的缩放比例。
- interpolation：指定插值。
- 扩展：OpenCV 提供的插值方式。
  - INTER_NN：最近邻插值。
  - INTER_LINEAR：双线性插值。
  - INTER_CUBIC：双三次插值。
  - INTER_AREA：区域插值。
  - INTER_LANCZOS4：兰索斯插值。

(3) cv2. flip 是 OpenCV 库中的一个函数，用于翻转图像。

```
cv2.flip(src,dst,flipcode)
```

- src：输入，原图，即待翻转的图像。
- dst：输出，改变后的图像。
- flipcode ：1，表示水平翻转；0，表示垂直翻转；-1，水平垂直翻转。

(4) cv2.getRotationMatrix2D 是 OpenCV 库中的一个函数，用于获取旋转矩阵。

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

- M：表示变换矩阵。
- corner：表示旋转中心。
- angle：旋转角度 (逆时针)。
- scale ：旋转后的缩放因子。

(5) cv2.getAffineTransform 是 OpenCV 库中的一个函数，用于获取仿射变换矩阵。。

```
M=cv2.getAffineTransform(pts1, pts2)
```

- M：表示变换矩阵。
- pts1：原图中三个点的坐标。
- pts2：原图中三个点在变换后相应的坐标。

4.3 几何变换示例

几何变换示例如下。

4.3.1 实验准备

本章的实践所使用的硬件和软件环境请参照第 1 部分实 践环境部分进行配置。

4.3.2 常用几何变换实例

本实验所需文件：image_transforms.py

本实验依赖库：OpenCV-Python (即 cv2) 和 NumPy

4.3.2.1 代码实现

代码实现如下。

```
#!/usr/bin/env python3
# encoding:utf-8
import cv2 as cv
import numpy as np
def main():
    # 读取图像
    im = cv.imread('lena.jpg')
    cv.imshow('lena.jpg', im)
    # 缩放图像
    dim = (int(im.shape[1]*1.3), int(im.shape[0]*1.3))
    im_rs_nr = cv.resize(im, dim, interpolation=cv.INTER_NEAREST)
    im_rs_ln = cv.resize(im, dim, interpolation=cv.INTER_LINEAR)
    im_rs_cb = cv.resize(im, dim, interpolation=cv.INTER_CUBIC)
    im_rs_lz = cv.resize(im, dim, interpolation=cv.INTER_LANCZOS4)
    cv.imshow('lena_rs_nr.jpg', im_rs_nr)
    cv.imshow('lena_rs_ln.jpg', im_rs_ln)
    cv.imshow('lena_rs_cb.jpg', im_rs_cb)
    cv.imshow('lena_rs_lz.jpg', im_rs_lz)
    # 沿 y 轴翻转图像 ( 水平翻转 )
    im_flip = cv.flip(im, 1)
    cv.imshow('lena_flip.jpg', im_flip)
    # 以图像中心为旋转点旋转图像
```

```
    h, w = im.shape[:2]
    M = cv.getRotationMatrix2D((w/2, h/2), 45, 1)
    im_rt = cv.warpAffine(im, M, (w, h))
    cv.imshow('lena_rt.jpg', im_rt)
    # 沿 x 轴负方向移动 100 个像素
    x = -100
    y = 0
    M = np.float32([[1, 0, x],[0, 1, y]])
    im_trans = cv.warpAffine(im, M, (w, h))
    cv.imshow('lena_trans.jpg', im_trans)
    cv.waitKey()
    cv.destroyAllWindows()
if __name__ == '__main__':
    main()
```

4.3.2.2 运行示例

我们可以按照下面的步骤进行实践。

(1) 首先按照第 I 部分要求进行硬件和软件环境配置，如果环境已经配置，本步可以跳过。

(2) 通过 cd 指令进入到存放有 image_transformer.py, 的文件目录下 (假定文件按照第 I 部分的路径组织)，该文件目录处于 Ubuntu 系统的桌面中的 examples 母文件夹中的子文件夹 02, 如图 4-2 所示。实际操作中读者可根据具体文件所在位置进入对应的路径下)。

图 4-2 进入指定路径

(3) 使用 python3 命令运行 image_transforms.py 文件，如图 4-3 所示。

图 4-3　运行相应文件

(4) 实验结果。

在 image_transforms.py 代码里，通过 x =–100，y = 0，M = np.float32([[1, 0, x],[0, 1, y]]) 来对 M 进行赋值，再通过 OpenCV 的库函数 cv.warpAffine(im, M, (w, h)) 来实现平移这一几何变换，结果如图 4.4(b) 所示，当然，我们可以修改 M 的值来实现我们所需的几何变换，具体知识请参考 4.2.2 节，例如令 x = 100，y = –100 时，我们得到结果 4.4(c)。

通过 OpenCV 的库函数 M = cv.getRotationMatrix2D((w/2, h/2), 45, 1) 来得到以 (w/2, h/2) 为变换中心，逆时针旋转 45° 的变换矩阵 ，再通过 OpenCV 的库函数 cv.warpAffine(im, M, (w, h)) 来实现旋转这个几何变换，如图 4.4(d) 所示，当我们将第三个参数改为 60 时，结果如图 4.4(e) 所示，cv.getRotationMatrix2D 库函数学习可参考 4.2.4 节。

通过 OpenCV 的库函数 cv.flip(im, –1)，由 4.2.4 节的介绍可知，可以实现原图的水平垂直变换，结果如图 4.4(f) 所示。

通过 dim = (int(im.shape[1]*1.3), int(im.shape[0]*1.3)) 将图片的宽和高都扩大为原图的 1.3 倍，通过 OpenCV 的库函数 cv.resize(im, dim, interpolation=cv.INTER_NEAREST) 来实现图片的缩放，第三个参数为采取的插值方式，不同插值方式的参数选择请参考 4.2.4 节，不同插值方式的具体介绍请参考 4.2.3 节。结果如图 4.4(g) 到图 4.4(j) 所示，请注意不同插值方式的区别。

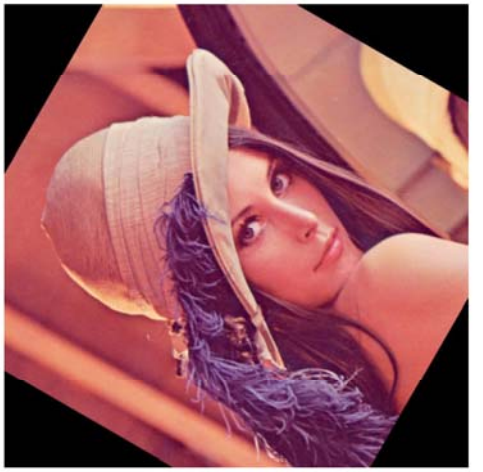
(a) 原图

(b) x 负方向平移 100 像素

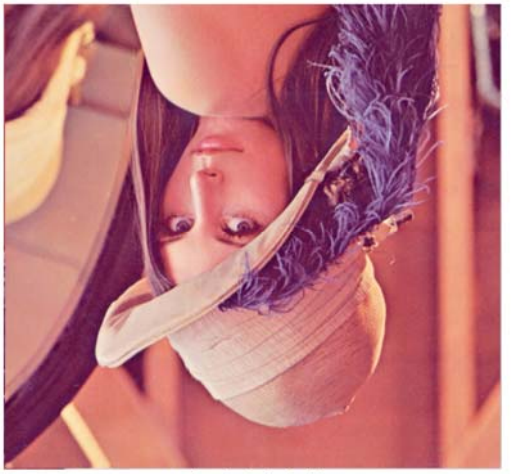
(c) x 正方向平移 100 像素，y 负方向平移 100 像素

(d) 逆时针旋转 45°

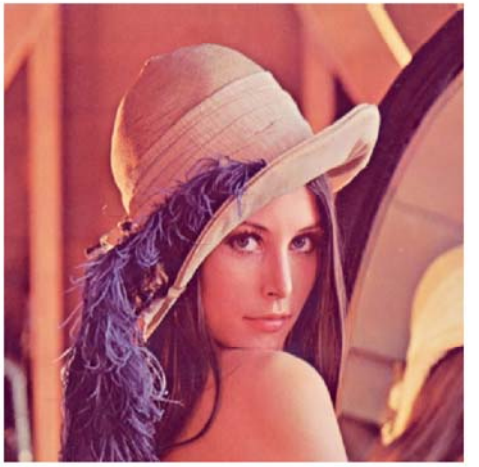
图 4-4　几何变换运行结果



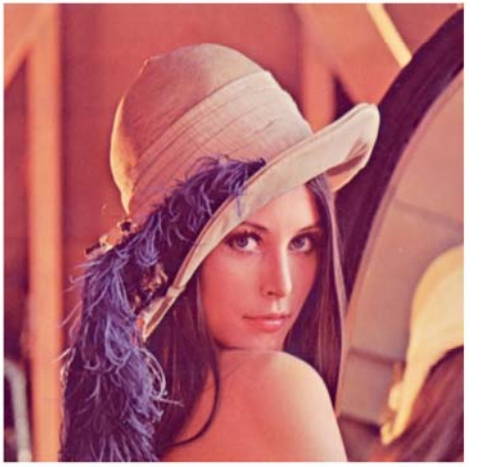
(e) 逆时针旋转 60°



(f) 水平垂直翻转



(g) 最近邻插值



(h) 双线性内插值

图 4-4　几何变换运行结果（续）

(i) 三次卷积插值

(j) 兰索斯插值

图 4-4 几何变换运行结果（续）

4.3.3 计算仿射变换矩阵

本实验所需文件：image_affine.py

本实验依赖库：OpenCV-Python（即 cv2）和 NumPy

4.3.3.1 代码实现

代码实现如下。

```
#!/usr/bin/env python3
# encoding:utf-8
import cv2 as cv
import numpy as np
def main():
    # 读取图像
    im = cv.imread('lena.jpg')
    cv.imshow('lena.jpg', im)
```

```
    # 用原始图像和目标图像中三对对应点计算仿射变换矩阵
    srcTri = np.float32([[0, 0], [im.shape[1] - 1, 0], [0, im.shape[0] - 1]])
    dstTri = np.float32([[0, im.shape[1] * 0.3], [im.shape[1] * 0.9, im.shape[0] * 0.2], [im.
shape[1] * 0.1, im.shape[0] * 0.7]])
    warp_mat = cv.getAffineTransform(srcTri, dstTri)
    # 对原始图像进行仿射变换得到目标图像
    im_affine = cv.warpAffine(im, warp_mat, (im.shape[1], im.shape[0]))
    cv.imshow('im_affine.jpg', im_affine)
    cv.waitKey()
    cv.destroyAllWindows()
if __name__ == '__main__':
    main()
```

4.3.3.2 运行示例

在 4.3.2 节实验的基础上，可以直接进行下面的步骤。如果设备断电关机，则需要参考前面实验的运行示例进行操作。

我们可以按照下面的步骤进行实践。

(1) 首先按照第 I 部分要求进行硬件和软件环境配置，如果环境已经配置，本步可以跳过。

(2) 通过 cd 指令进入到存放有 image_affine.py, 的文件目录下 (假定文件按照第 I 部分的路径组织)，该文件目录处于 Ubuntu 系统的桌面中的 examples 母文件夹中的子文件夹 02, 如图 4-5 所示。实际操作中读者可根据具体文件所在位置进入对应的路径下) 。

图 4-5 进入指定路径

(3) 使用 python3 命令运行 image_affine.py 文件，如图 4-6 所示。

```
khadas@Khadas: ~/Desktop/examples/02
khadas@Khadas:~$ cd Desktop
khadas@Khadas:~/Desktop$ cd examples
khadas@Khadas:~/Desktop/examples$ cd 02
khadas@Khadas:~/Desktop/examples/02$ python3 image_affine.py
```

图 4-6　运行相应文件

(4) 实验结果。

在 image_affine 代码中，我们通过 dstTri = np.float32([[0, im.shape[1] * 0.3], [im.shape[1] * 0.9, im.shape[0] * 0.2], [im.shape[1] * 0.1, im.shape[0] * 0.7]]) 来定义变换后图像的三个点坐标，再通过 OpenCV 的库函数 cv.getAffineTransform(srcTri, dstTri) 来获取相对应的 M 的值，最后我们通过 OpenCV 的库函数 cv.warpAffine 来实现原图到指定图片位置的变换，如图 4-7 所示。

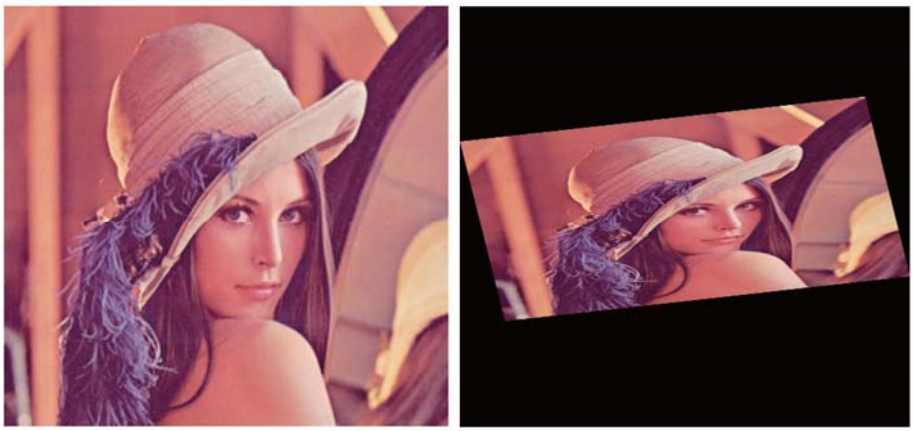


图 4-7　仿射变换运行结果

4.4　小结

图像的几何变换是图像处理中的重要操作之一，它可以对图像进行平移、旋转、缩

放和倾斜等变换，从而改变图像的位置、尺寸和形状。通过几何变换，我们可以实现图像的校正、对齐、变形和投影等操作。

在几何变换中，平移操作可以将图像沿着水平和垂直方向移动，改变图像的位置。旋转操作可以围绕一个指定的中心点对图像进行旋转，使图像在角度上发生改变。缩放操作可以按比例调整图像的尺寸，使其更大或更小。倾斜操作可以使图像在水平或垂直方向上倾斜，改变图像的形状。除了这些基本的几何变换，还有更复杂的仿射变换和透视变换，它们可以实现更灵活和精确的图像变换。

几何变换在计算机视觉、图像匹配、图像配准和增强现实等领域具有广泛的应用。通过平移和旋转，我们可以对图像进行对齐和校正，以便进行特征提取和匹配。缩放操作可以用于图像的放大或缩小，以适应不同的显示或分析需求。倾斜操作可以用于纠正图像中的透视畸变，使其符合真实场景的几何形状。

要进行图像的几何变换，我们可以使用线性代数和几何变换矩阵来描述和实现变换过程。通过应用适当的变换矩阵，我们可以对图像进行精确的变换操作，并获得所需的结果。

总而言之，图像的几何变换是一种强大的工具，可以改变图像的位置、尺寸和形状，以适应不同的需求和应用场景。它在图像处理和计算机视觉中扮演着重要角色，为我们提供了丰富的图像处理和分析手段。

4.5 实践习题

(1) 用 OpenCV 提供的函数实现其余几何变换。

(2) 讨论仿射变换在现实中的应用 (提示：图像拼接)。

(3) 请用 OpenCV 实现将图像放大 1.2 倍再以图像中心为旋转点顺时针旋转 30 度，然后沿 x 轴方向移动 50 个像素的操作，并输出结果图像 (见图 4-8)。

图 4-8　运行结果