实验：NumPy 转换

实验概要

如果您曾经看过电影的“幕后花絮”视频，可能已经注意到某些场景是在绿幕前拍摄的。然后，在编辑过程中，将绿幕替换为令人难以置信的未来派远景或反乌托邦场景，这超出了您的想象。现在，考虑电影中该场景的一帧。原始帧是在绿色背景前拍摄的。然后，对图像进行修改以获得特定的结果，即不同的背景。此外，有时，您希望处理图像以直接获得所需的结果，以便获得中间结果，这将使进一步的步骤变得容易且可实现。以下是在绿色屏幕前拍摄的示例图片：

修改或处理图像以获得特定结果的这种活动，称为 图像处理 。在上一章中，我们介绍了图像的基础知识：什么是像素，什么是像素坐标，如何使用像素坐标提取像素值等等。本章我们将通过理解图像处理的含义以及为什么需要它来开始。到最后，您将能够处理诸如上图所示的图像，并使用称为 掩码 的非常基本的图像处理技术将绿屏替换为您选择的背景。

本章可以分为两个主要部分 ——

- 前半部分，我们将专注于基本技术，例如平移，旋转，调整大小和裁剪，以及更通用的几何变换（仿射变换）、透视变换。
- 后半部分，我们将讨论二进制图像和可以对图像执行的算术运算。

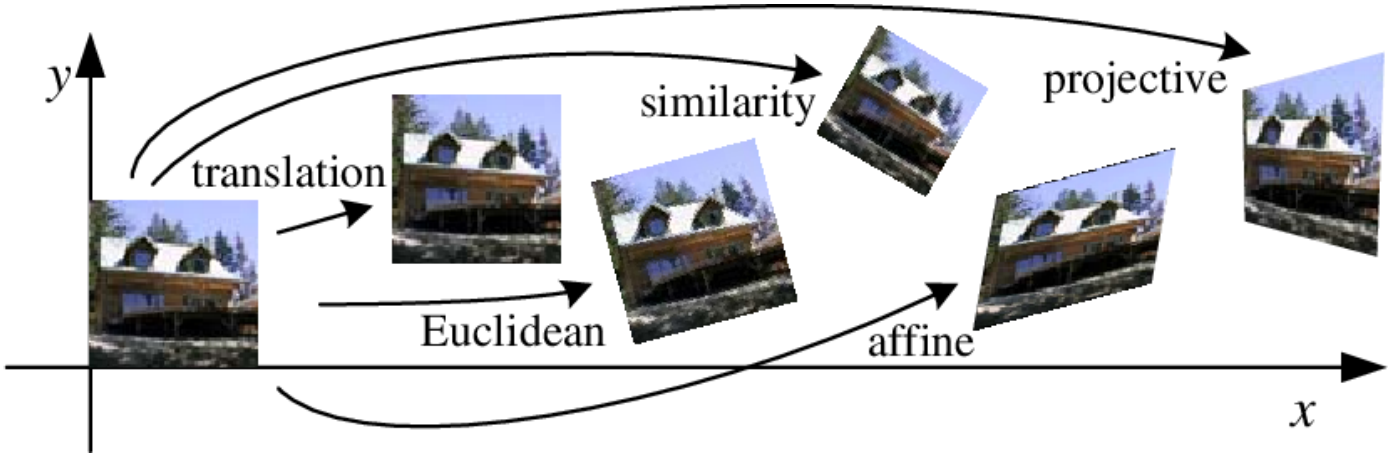
几何变换（Geometric Transformations）

通常，在图像处理期间，需要转换图像的几何形状（例如：图像的宽度和高度）。此过程称为 几何变换 。正如我们之前所看到的，图像不过是 矩阵 ，因此我们可以使用一种更数学的方法来理解这些主题。由于图像是矩阵，因此，如果我们对图像（矩阵）进行运算，最后得到另一个矩阵，则称此为 变换 。该基本思想将广泛用于理解和应用各种几何变换。

这是我们将在后面的实验中讨论的几何变换：

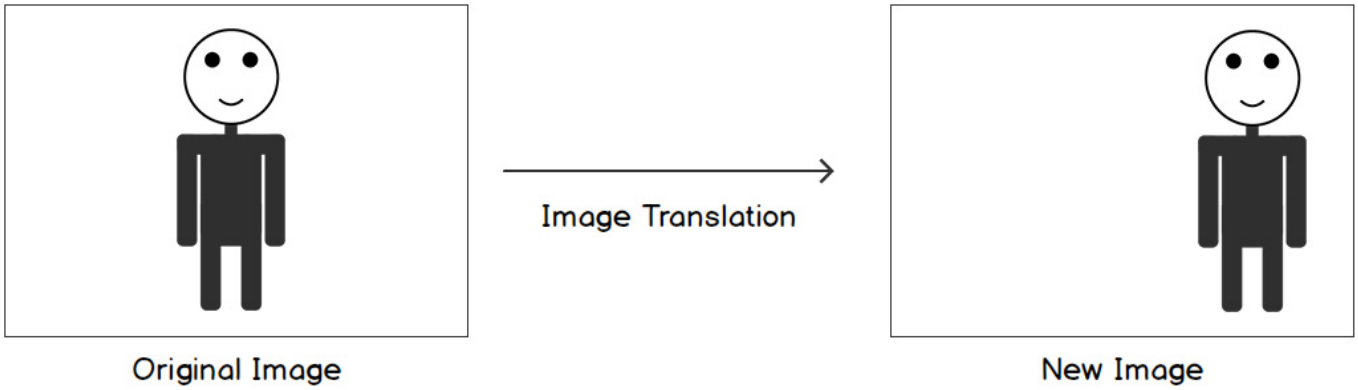
- **平移**
- **旋转**

- **图像缩放**
- **仿射变换**
- **透视变换**

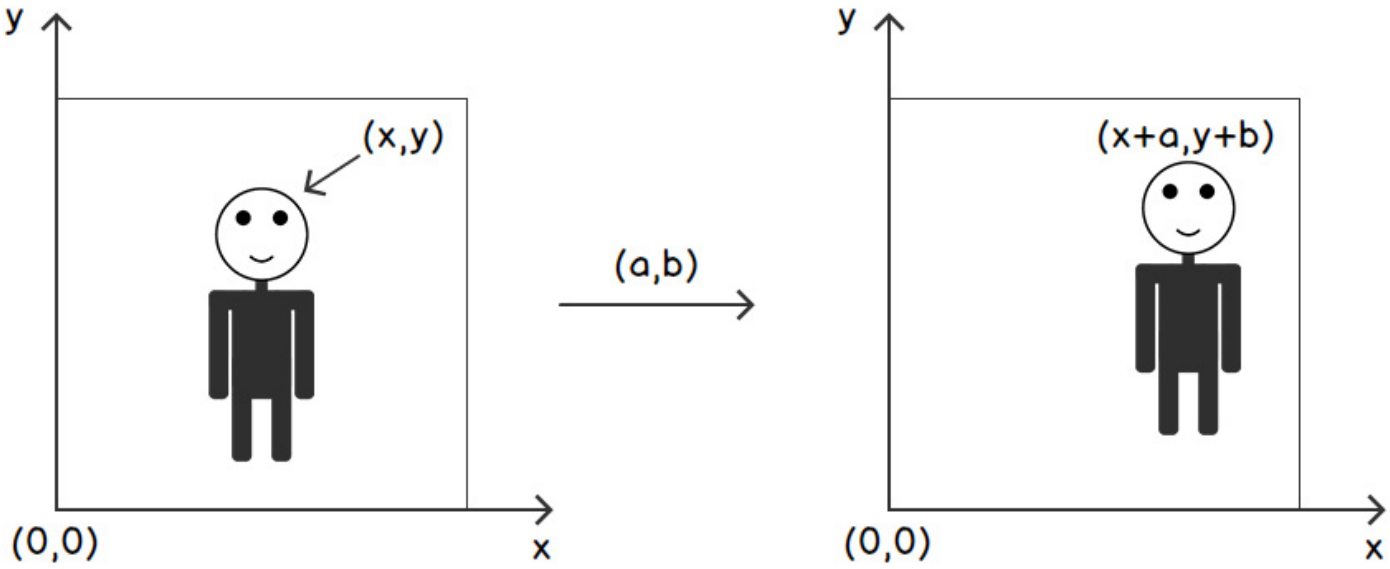


图像转换（Image Translation）

jfif 图像平移也称为移动图像。在这个转换中，我们的基本意图是沿着一条线移动图像。请看下图：



例如，如果我们考虑下图，我们正在将人体素描向右移动。我们可以在 X 和 Y 两个方向移动图像，也可以分别移动。在这个过程中，我们要做的是将图像中的每个像素的位置向一个特定的方向移动。



手动计算

使用上图作为参考，我们可以使用以下矩阵方程式表示图像平移：

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix}$$

在前面的等式中， x’ 和 y’ 代表点/像素在 X 方向上移动了 a 个单位和在 Y 方向上移动了 b 个单位后的新坐标。

转换矩阵

现在，我们已经了解了图像转换的基本理论，让我们了解如何使用 NumPy 进行转换。在 Python 的 OpenCV 中使用图像时，图像不过是 NumPy 数组。同样，几何变换只是矩阵乘法。我们还看到了如何使用矩阵方程表示图像平移。对于矩阵乘法，相同的图像方程式可以写为 NumPy 数组：

```
M = np.array([[1,0,a],[0,1,b],[0,0,1]])
```

您会注意到转换矩阵 ——

- 前两列：构成一个单位矩阵（[[1,0],[0,1]]）。 这表示即使我们正在变换图像，图像的尺寸（宽度和高度）也将保持不变。
- 最后一列：由 a 和 b 组成，这表示图像已在 X 方向上移动了 a 单位，在 Y 方向上移动了 b 单位。
- 最后一行： [[0,0,1]] 仅用于制作矩阵 M，即正方形矩阵 —— 包含相同行数和列数的矩阵。

现在我们有了用于图像转换的矩阵，每当我们要移动图像时，我们要做的就是将图像 img （作为 NumPy 数组）乘以数组 M 。 执行如下：

```
output = M@img
```

从而，获得图像转换后的输出 —— 下图稍微复习矩阵乘法过程：

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} j & k & 1 \\ m & n & 0 \\ p & q & r \end{bmatrix} =$$

$$\begin{bmatrix} aj+bm+cp & ak+bn+cq & al+bo+cr \\ dj+em+fp & dk+en+\text{fq} & dl+e0+\text{fr} \\ gj+hm+ip & gk+hn+\text{iq} & gl+ho+ir \end{bmatrix}$$

请注意，在前面的代码中，假设我们要移动的每个点都代表矩阵中的一列，即 img 。通常，在 img 矩阵中还会添加一个满是 1 的额外行。这仅仅是为了确保可以进行矩阵乘法。为了理解它，让我们了解矩阵 M 的尺寸 —— 它具有 3 行 3 列。为了执行 M 和 img 的矩阵乘法， img 的行数应与矩阵M中的列数相同，即为 3 。

现在，我们还知道一个点在 img 矩阵中表示为一列。我们已经在 img 矩阵的第一行中获得了该点的 X 坐标，并在第二行中获得了该点的 Y 坐标。为了确保 img 矩阵中有三行，我们在第三行中加 1 。

例如，点 [2,3] 将表示为 [[2],[3],[1]] ，即第一行中存在 2 ，第二行中存在 3 。第三行是 1 。

同样，如果我们要表示两个点： [2,3] 和 [1,0] ，它们将表示为： [[2,1],[3,0],[1,1]] ；也就是说，第一行中同时存在两个点（ 2 和 1 ）的 X 坐标，第二行中存在 Y 坐标（ 3 和 0 ），第三行全部 1 组成。

让我们借助实验操作更好地理解这一点。

实验目标

在本实验中，我们将使用 NumPy 进行称为平移的几何变换。

我们将使用我们之前看过的转换矩阵来进行转换。我们将 3 个点在 X 方向上移动 2 个单位，在 Y 方向上移动 3 个单位。

1. 导入 NumPy 模块

In [1]:

```
# 导入NumPy模块
import numpy as np
```

2. 指定要移动的点的坐标值

接下来，我们将指定三个想要移动的点: [2,3]， [0,0] 和 [1,2]。正如我们之前看到的，这些点将表示为:

In [2]:

```
# 指定要移动的3个点的坐标值
points = np.array([[2,0,1],        # 分别指定每个点的 X 坐标
                   [3,0,2],        # 分别指定每个点的 Y 坐标
                   [1,1,1]])       # 全 1 填充
```

注意， X 坐标 (2,0 和 1) 组成第一行， Y 坐标 (3,0 和 2) 组成第二行，而第三行只有 1 。

In [3]:

```
#  输出点的矩阵
print(points)
```

```
[[2 0 1]
 [3 0 2]
 [1 1 1]]
```

3. 手动计算转换后的坐标

In [4]:

```
# 使用手动计算定义输出点outPoints
outPoints = np.array([[2 + 2, 0 + 2, 1 + 2],  # 所有点在X轴移动2个单位，因此分别+2
                      [3 + 3, 0 + 3, 2 + 3],  # 所有点在Y轴移动3个单位，因此分别+3
                      [1, 1, 1]])             # 全 1 填充行不变
```

In [5]:

```
#  输出outPoints的矩阵
print(outPoints)
```

```
[[4 2 3]
 [6 3 5]
 [1 1 1]]
```

4. 定义转换矩阵

同样，我们需要在 X 方向上移动 2 个单位，在 Y 方向上移动 3 个单位，定义转换参数 a 和 b：

In [6]:

```
# 在X方向上移动2个单位
a = 2

# 在Y方向上移动3个单位
b = 3
```

In [7]:

```
# 定义转换矩阵M
M = np.array([[1,0,a],[0,1,b],[0,0,1]])
```

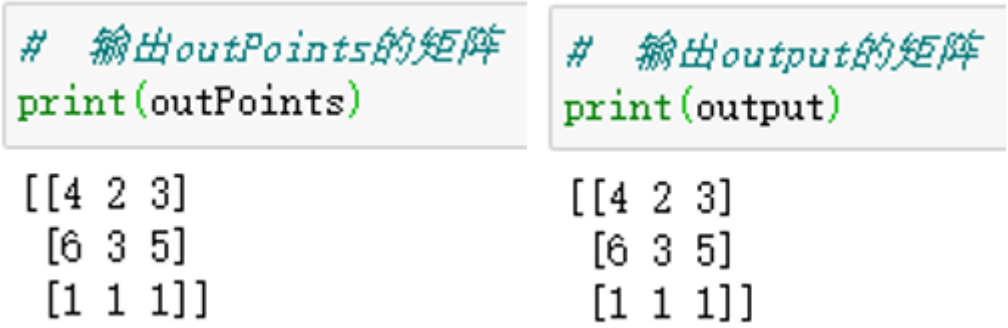
In [8]:

```
# 输出转换矩阵M
print(M)
```

```
[[1 0 2]
 [0 1 3]
 [0 0 1]]
```

5. 使用矩阵乘法进行平移

通过下面的代码可以看到，通过转换矩阵与原始点 points 执行矩阵乘法，我们可以得到与手动计算出的平移后的坐标矩阵 outPoints 相同的结果 output



In [9]:

```
# 使用NumPy执行转换
output = M@points
```

In [10]:

```
# 输出output的矩阵
print(output)
```

```
[[4 2 3]
 [6 3 5]
 [1 1 1]]
```

实验小结

在本实验中，你可以看到通过 NumPy 矩阵乘法和使用手动计算所获得的输出点坐标结果是如何完美匹配的。我们使用手工计算和矩阵乘法来移动给定的点。我们还看到了如何使用 Python 中的 NumPy 模块来执行矩阵乘法。这里要关注的关键点是：转换只是一个矩阵运算操作。