计算机视觉实验指导书

实验二：SIFT特征点与图像拼接

华为智能基座课程



广东工业大学

计算机学院

2024年2月

**1. 实验介绍**

实验前理论课应该完成图像特征和图像拼接部分的讲解，学生已了解、掌握常见图像处理的基本原理，以及python语言的基本语法和使用。本实验首先回顾相关知识点，然后通过具体的案例实现基于SIFT特征点的图像拼接算法。

**2. 知识回顾**

2.1 图像拼接概述

在拍摄集体照时，如果一次拍照不足以把所有人都排到，就需要在不同位置拍多张照片，然后将这些照片拼接起来。如图2.1所示，**图像拼接涉及的内容包括：特征点提取、特征点匹配、图像扭曲等**。

现在考虑最简单的情况：两幅待拼接的图像仅有位置的平移。此时，仅需在两幅图像中各找一个相同的点（比如：建筑物的尖角），然后将这两个点对齐，就能实现图像的拼接。然而，两次拍摄时的相机不仅存在位置的差异，还可能存在旋转、拍摄角度、甚至时间的差异。这样，为了达成图像拼接的目的，需要选择更多的两幅图像中能对应的点。这种能用于图像拼接的、两幅图像上都有的对应点，叫做**特征点**。当等待拼接的图像数量很多时，手动选择特征点就显得费时费力了。因此，人们开发了许多全自动的特征点提取算法。本任务选择SIFT（Scale Invariant Feature Transform，尺度不变特征变换）作为特征点的自动提取算法。

在两幅图像中提取到各自的特征点，还需要将这些点对应起来，这样才能根据匹配的点进行图像扭曲和融合。能相互匹配的特征点，对应两幅图像中相同的物理位置。若用一个“**特征向量**”来描述特征点，那么能相互匹配的特征点的特征向量，应该是相似的。实践中，我们通过计算两个特征向量的欧式距离来判断相似性。

相片是将三维空间的信息投影在二维平面上得到的。根据相机的位置、角度不同，拍出的相片间存在诸如平移、旋转、斜切、畸变等差异。现在有了相互匹配的特征点，我们根据这些匹配关系，可确定两幅待拼接图像之间的变换。这种变换可用一个**单应性矩阵**来描述。因此，图像拼接的最后一步，就是根据成对的特征点，确定变换矩阵，然后将其中一幅图像变换到另一幅图像的二维坐标系内。

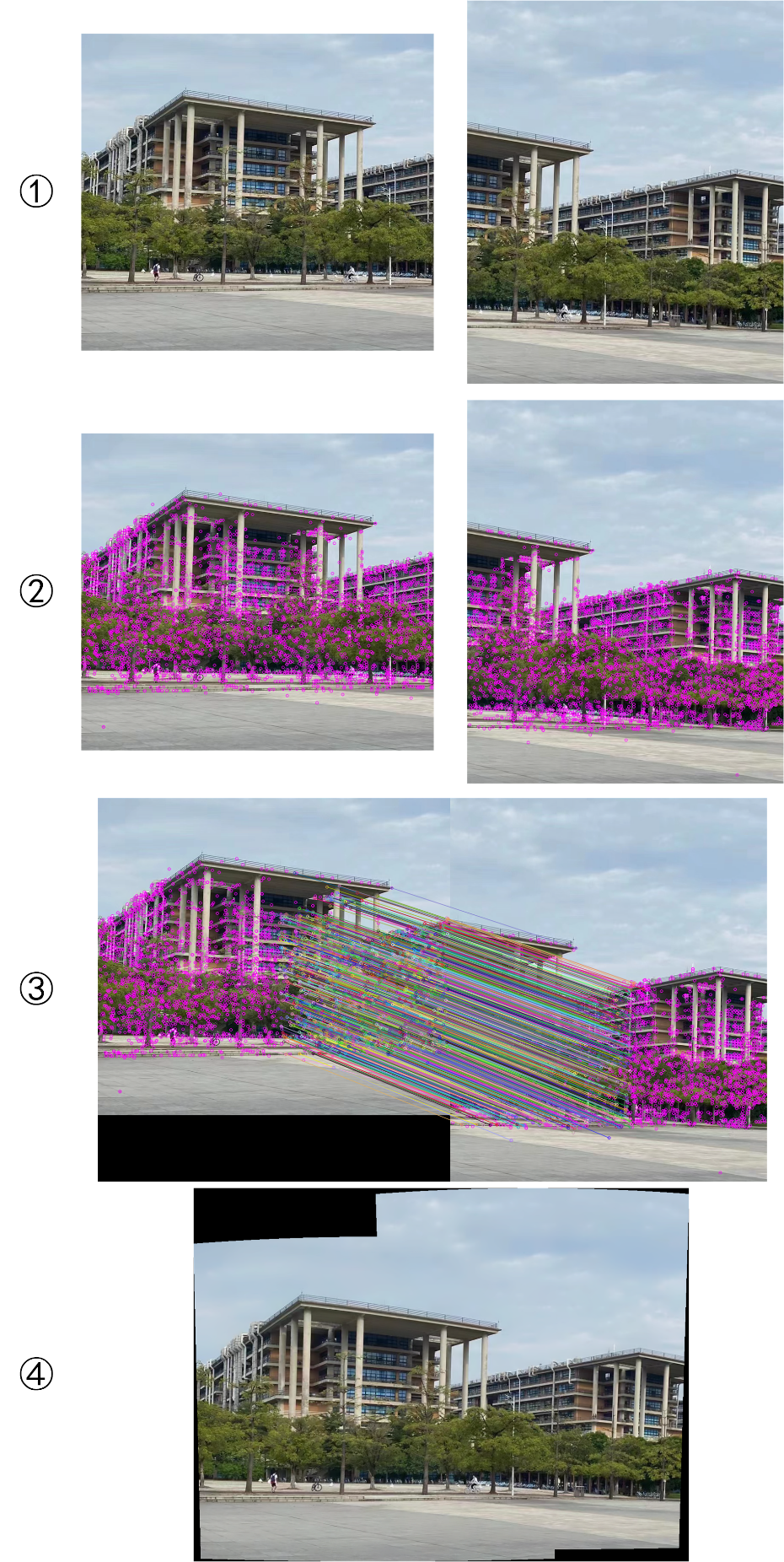


图2.1 图像拼接的一般流程。①待拼接图像获取。②特征点提取。③特征点匹配。④基于匹配点的图像形变与融合。

2.2 特征点选择标准

“特征点”中的“特”字，已经说明了这个点应该是独特的、有区分度的。这样，两幅图像中才能找到一样的配对点。图像的局部极大值或极小值点，是一种简单而有效的选择特征点的思路。因为我们要在两幅图像寻找成对的特征点，那么图像的缩放、旋转、形变、光照变化，不应影响特征点的提取。换句话说，特征点应该是稳定的。最后，作为一种实用的程序，特征点的计算应该是高效的。总结起来，一个好的特征点应该具备如下性质：

* + 独特、有区分度；
  + 对图像变形和光照变化稳定；
  + 容易计算。

2.3 SIFT特征点提取

SIFT算法选择图像的局部极值点作为特征点。由于图像中有区分度的点所对应的实际物体是有大有小的，SIFT为了寻找到各种有价值的特征点，首先创建这幅图像的多尺度高斯差分金字塔。

多尺度金字塔即对原始图像进行多次缩小，形成一个金字塔形的数据结构，如图2.2所示。这样，缩小后图像上的大目标，变得与原图像上的小目标一样大。这将允许算法使用同一种尺寸的检测器检测所有有价值的目标。

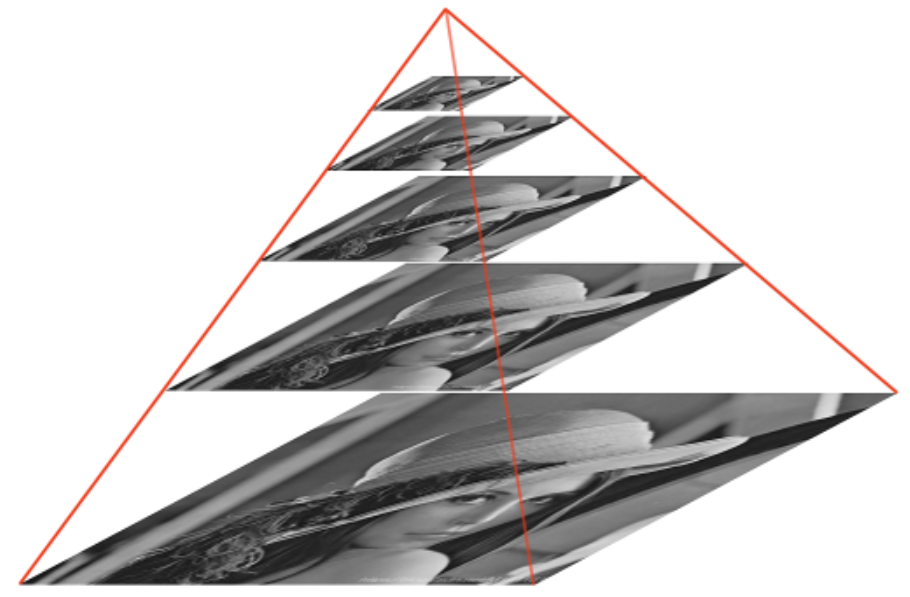


图2.2 多尺度金字塔示意图。

SIFT算法在原始图像的边缘图上寻找局部极值点，而并非在原始图像上寻找。边缘图能更好地描述目标的形状，可以看作对原始图像的信息提纯。SIFT算法通过高斯差分的方式计算待拼接图像的边缘图：

，

其中，I和edge分别是输入图像和边缘图像，表示对*I*进行高斯平滑滤波，其公式为：

，

其中，控制模糊的程度，越大，模糊效果越明显。相当于图像的低频成分，对应灰度变换不大的均匀色块。将低频成分从原图中去除，得到的是图像的高频成分，也就是目标的边缘。

对多尺度金字塔的每一层，SIFT算法对金字塔以不同的参数执行多次高斯模糊，从而得到多张差分图像。金字塔的每一层均产生5张高斯模糊图像，其参数分别为：。

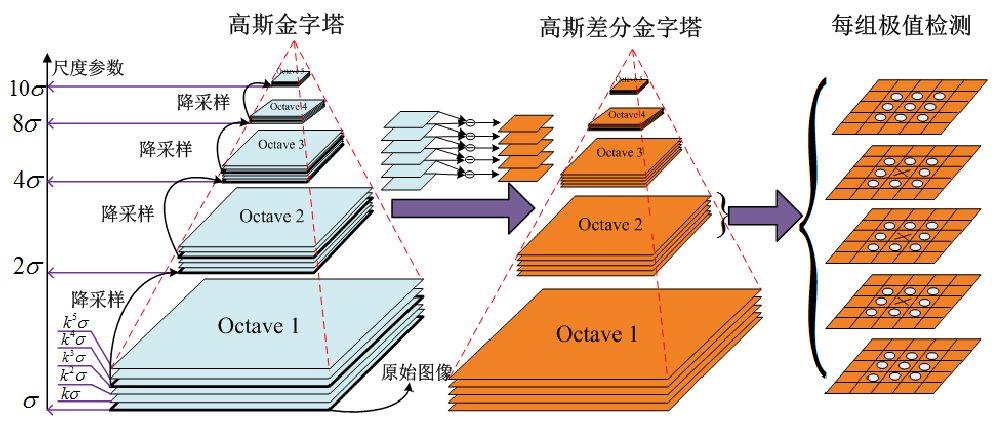


图2.3 高斯差分金字塔示意图。

在高斯差分金字塔上选择局部极值点。对于一个像素，考察它的8-邻域以及上下两层中3行3列的像素点（共计个）的灰度值。如果这个像素点的灰度值比另外26个临近点都大/小，说明该点是一个局部极大/小值点。找到极值点后，还应对其进行筛选。因为图像是对现实物体的离散采样，通过上述方法找到的极值点，可能并不是连续空间的极值点，而是一个灰度值非常高/低的噪声点。为此，SIFT算法会在候选的极值点处进行曲线拟合，将局部曲率非常不对称的点删除。这样，完成了用于图像拼接的特征点的提取。

2.4 SIFT特征向量计算

为了进行图像拼接，需要将两幅待拼接图像里的特征点进行配对。此处，我们通过“特征向量”这一概念来描述一个特征点的特点。对于两个能相互匹配的特征点，它们所对应的物理位置是相同的，它们的特征向量也应该是相似的。通过计算两个向量的欧氏距离，我们可以判断两个特征点的相互匹配的程度。

SIFT算法对特征向量的计算分两个步骤。首先，寻找特征点位置的“主方向”。如图2.4所示，“主方向”就是特征点附近一个半径为的圆形邻域内，统计最常出现的梯度方向。在统计时，邻域中每个位置的梯度值会附加一个高斯系数，从而使距离特征点近的梯度权重更大。这个高斯系数的方差设置为。找到主方向后，将坐标轴旋转至主方向，重新采集特征点附近的梯度信息，如图2.5所示。按主方向将坐标轴旋转，可减少SIFT特征向量对方向的敏感度，从而成为一种更加鲁棒的特征。

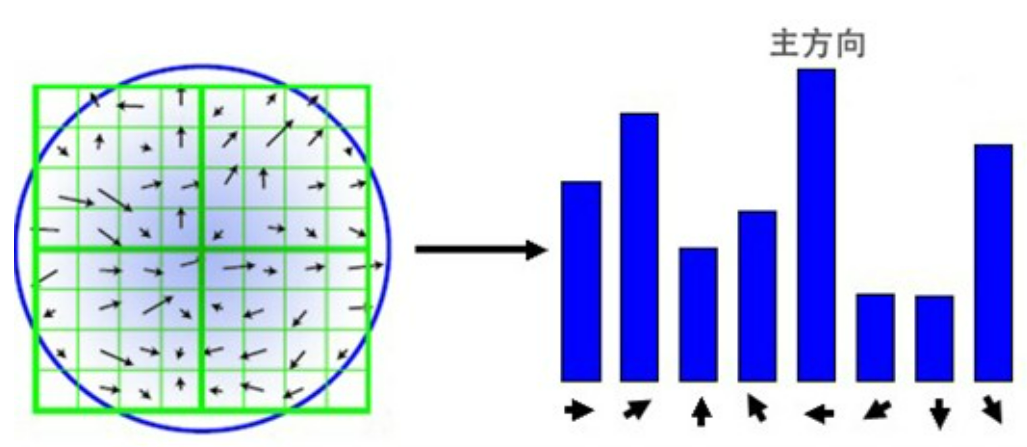


图2.4 SIFT算法计算特征向量的第一步：按特征点附近的梯度信息，将坐标轴旋转至主方向。

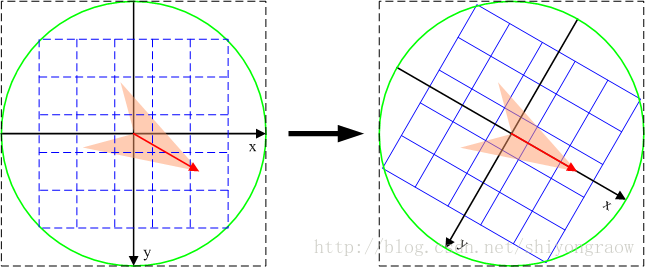


图2.5 将坐标轴旋转至主方向

第二步，统计特征点附近的梯度信息，将其整合为一个特征向量。如图2.12所示，对于一个方格，我们统计8个方向的梯度值。对每个特征点，SIFT统计附近16个方格的梯度信息。因此，SIFT特征向量的长度为。

2.5 SIFT特征点匹配与图像拼接

通过特征向量的欧式距离进行特征点匹配。特征空间内两个特征向量的欧式距离，等于向量差的模长。为了找到图像1中特征点在图像2中的匹配点，计算特征向量与图2中所有特征向量的距离。最后，选择距离最近的且距离小于一定预设长度的特征点，作为的匹配点。实际应用中，由于穷举匹配的计算效率较低，一般采用kd树算法来寻找相互匹配的特征点。

根据相互匹配的特征点，计算单应性矩阵。进而将待匹配的图像2进行位移、旋转、缩放、拉伸、扭曲等操作，将其转换到与图像1相同的坐标轴上。最后，将两幅图像相同的部分对齐，实现图像拼接。

**3. 上机实验**

备注：

1. 部分路径需按自己电脑进行设置，部分代码（红色）需要学生补全。

2. 压缩包内的图片仅供参考。学生需使用自己拍摄的图片完成3.1~3.4的基础实验，将运行结果和必要说明写在实验报告中。

3. 结合3.1~3.4的代码，完成思考题。应给出源代码、运行结果。

3.1 两幅图像拼接

读取待匹配的两幅图像，将其合并为一个数组。随后，直接调用OpenCV的Stitcher类实现图像拼接。Stitcher类整合了上述特征点提取、特征向量计算、特征点匹配和图像形变等操作。

import os

import sys

import numpy as np

import cv2 as cv

import matplotlib.pyplot as plt

os.chdir('pwd\\')

# imgPath为图片所在的文件夹相对路径

imgPath = 'sift\\'

imgList = os.listdir(imgPath)

imgs = []

for imgName in imgList:

pathImg = os.path.join(imgPath, imgName)

img = cv.imread(pathImg)

if img is None:

print("图片不能读取：" + imgName)

sys.exit(-1)

# append函数能将多个数组拼接起来

imgs.append(img)

# 调用Stitcher类，执行图像拼接

stitcher = cv.Stitcher.create(cv.Stitcher\_PANORAMA)

\_result, img12 = stitcher.stitch(imgs)

# 保存拼接结果

cv.imwrite('result\\p12.png', img12)

3.2 单独计算图像的高斯差分图像

# 原始图像读取。此处将图像类型变为double，目的方便未来存储负数的灰度值

im1 = cv.imread('sift\\p1.png')

im1 = (im1[:,:,0]).astype(np.double)

# 首先对原始图像进行高斯模糊

sz = 7

sig = 3

# 自行查找 cv.GaussianBlur 的使用方法，完成高斯模糊

# im\_gs =

# 执行高斯差分。即原始图像减去模糊后的图像

im3 = im1 – im\_gs

# 对高斯差分图像进行灰度值归一化

# 将 im3 的灰度值归一化至[0,255]

# im3 =

# 保存高斯差分图像

cv.imwrite('result\\s1-DOG.png', im3)

3.3 单独提取SIFT特征点和特征向量

此处通过OpenCV提供的SIFT\_create()来实现。

sift = cv.SIFT\_create()

img1 = cv.imread('sift\\p1.png') # 读取两幅待匹配的图像

img2 = cv.imread('sift\\p2.png')

# 获取每一幅图像的特征点及SIFT特征向量

# 返回值tp包含SIFT特征的方向、位置、大小等信息

# des的shape为（sift\_num， 128）， sift\_num表示图像检测到的SIFT特征数量

(tp1, des1) = sift.detectAndCompute(img1, None)

(tp2, des2) = sift.detectAndCompute(img2, None)

# 在原始图像上绘制特征点，用紫色圆圈表示

sift\_1 = cv.drawKeypoints(img1, tp1, im1, color=(255, 0, 255))

sift\_2 = cv.drawKeypoints(img2, tp2, im2, color=(255, 0, 255))

# 保存绘制结果

cv.imwrite('result\\sift\_1.png', sift\_1)

cv.imwrite('result\\sift\_2.png', sift\_2)

3.4 对两幅图像的特征点进行匹配

此处使用了OpenCV的BFMatcher类提供的K临近算法。

# 调用BFMatcher类，对特征向量集合des1和des2进行匹配

bf = cv.BFMatcher()

matches1 = bf.knnMatch(des1, des2, k=2)

# 调整ratio的数值。若对匹配精度要求较高，可适当降低ratio数值。

# 若待匹配的特征点数量较多，则可适当提高rario数值。

#一般情况下可设置ratio为0.5。

ratio1 = 0.5

good1 = []

# 如果最接近和次接近的比值大于一个预设的数值，那么保留这个最接近的值，并将其设置为匹配点

for m1, n1 in matches1:

if m1.distance < ratio1 \* n1.distance:

good1.append([m1])

# 将相互匹配的特征点用线连接起来，保存匹配结果

match\_result1 = cv.drawMatchesKnn(img1, tp1, img2, tp2, good1, None, flags=2)

cv.imwrite("result\\sift\_match.png", match\_result1)

思考题：

1. 使用Stitcher类，实现4张或以上图像的拼接。
2. 结合3.2的代码和图2.3，计算一幅图像的高斯差分金字塔。请展示金字塔每一层中 最底层 图像的 **高斯模糊**和**高斯差分**效果。（）
3. 将找到的极值点以彩色点标注在原始图像上（来自金字塔第一层的极值点-红色，第二层-绿色，第三层-蓝色，第四层-紫色，第五层-黄色）
4. 结合3.3和3.4的代码，计算 能够相互匹配的 特征点之间的平均欧式距离（特征向量之间的距离，而不是坐标位置的距离）

