撰写日期 Date: 2020-12-7

多媒体概论课程设计

课程名称： 多媒体概论

主讲教师： 徐媛媛

学 分 数： 2 Credits

开设学院： 计算机与信息学院 Computer and Information Science

开设时间： 2020 年第一学期 1st Semester, 2020

组员：

姓名	班级	学号
颜小涵	计算机 6 班	1806010621

成绩 Grade：

河 海 大 学

目 录

一、项目意义

全景图拼接技术是全国范围内迅速发展并逐渐流行的一种视觉技术，其是一种基于图像绘制技术生成真实感图形的虚拟现实技，全景图拼接技术主要是通过对图像的拼接，实现对场景的环视。全景图拼接技术本身是一种比较实用的技术，现在发展较快，在各类手机 app 和图像 api 中都包含这种技术。学习和实践全景图拼接技术有利于促进我们对 OpenCV 库的理解，加强我们对图像特征工程的认知与学习。

在这个项目之中，我将使用 OpenCV 库对图像进行预处理、采用 SIFT 进行关键特征点提取，使用 knn 检测来进行特征点匹配，最后使用视角变换矩阵进行翘曲变换和边缝融合完成多张图片的全景图拼接。

在自己完善部署整个拼接函数之后，我还尝试使用了 OpenCV 库自带的 cv2.createStitcher()函数进行全景图拼接，并输出两个图像，比较拼接效果的异同。

二、背景知识

(一) SIFT 算法

SIFT 算法名叫尺度不变特征转换算法，其是一种计算机视觉的算法。它主要用来侦测与描述影像中的局部性特征，它在空间尺度中寻找极值点，并提取出其位置、尺度、旋转不变量。局部影像特征的描述与侦测可以帮助辨识物体，SIFT 特征是基于物体上的一些局部外观的兴趣点而与影像的大小和旋转无关。对于光线、噪声、些微视角改变的容忍度也相当高。基于这些特性，它们是高度显著而且相对容易撷取，在母数庞大的特征数据库中，很容易辨识物体而且鲜有误认。使用 SIFT 特征描述对于部分物体遮蔽的侦测率也相当高，甚至只需要 3 个以上的 SIFT 物体特征就足以计算出位置与方位。在现今的电脑硬件速度下和小型的特征数据库条件下，辨识速度可接近即时运算。SIFT 特征的信息量大，适合在海量数据库中快速准确匹配。

SIFT 算法的实质是在不同的尺度空间上查找关键点，并计算出关键点的方向。SIFT 所查找到的关键点是一些十分突出，不会因光照，仿射变换和噪音等因素而变化的点。因此，SIFT 算法对于光线、噪声、些微视角的容忍度非常高，所以当我们需要拼接两张图片 A 和 B 时，我们可以使用 SIFT 算法分别提取出两张图片的所有特征点。

(二) KNN 算法

KNN 是通过测量不同特征值之间的距离进行分类。它的思路是：如果一个样本在特征空间中的 k 个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别，其中 K 通常是不大于 20 的整数。KNN 算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。KNN 算法就是在训练集中数据和标签已知的情况下，输入测试数据，将测试数据的特征与训练集中对应的特征进行相互比较，找到训练集中与之最为相似的前 K 个数据，则该测试数据对应的类别就是 K 个数据中出现次数最多的那个分类。

在全景图拼接中，我们将 K 定为 2 并使用 KNN 算法对左图像与右图像的特征点对进行分类。选择最接近的 2 个特征点进行特征点匹配。在匹配好的特征点对的基础上保留一定的特征点对，当剩余点对大于 4 的时候即可计算单应矩阵进行全景图拼接。

(三) RANSAC 算法

RANSAC 是一种迭代算法,用来从观测数据中估算出数学模型的参数，此基础上便可以分离内群与离群数据。简单来说就是一般来讲观测的数据里经常会出现很多噪音，比如说像 SIFT 匹配有时就会因为不同地方有类似的图案导致匹配错误。而 RANSAC 就是通过反复取样，也就是从整个观测数据中随机抽一些数据估算模型参数之后看和所有数据误差有多大，然后取误差最小视为最好以及分离内群与离群数据。RANSAC 可以鲁棒的估计模型参数。

在全景图拼接中，我们使用 RANSAC 算法来计算对应的单应矩阵。单应矩阵由两幅图像的对应点计算得来，每个对应点可以写出两个方程，分别对应 x 坐标和 y 坐标，因此计算单应矩阵 H 至少需要 4 对匹配点。每次从所有点对中取出 4 对，计算指定的单应矩阵，选出内点最多的单应矩阵作为最终的结果。

$$\begin{bmatrix} x_1, y_1, 1, 0, 0, 0, -x_1'x_1, -x_1'y_1 \\ 0, 0, 0, x_1, y_1, 1, -y_1'x_1, -y_1'y_1 \\ x_2, y_2, 1, 0, 0, 0, -x_2'x_2, -x_2'y_2 \\ 0, 0, 0, x_2, y_2, 1, -y_2'x_2, -y_2'y_2 \\ x_3, y_3, 1, 0, 0, 0, -x_3'x_3, -x_3'y_3 \\ 0, 0, 0, x_3, y_3, 1, -y_3'x_3, -y_3'y_3 \\ x_4, y_4, 1, 0, 0, 0, -x_4'x_4, -x_4'y_4 \\ 0, 0, 0, x_4, y_4, 1, -y_4'x_4, -y_4'y_4 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix} \qquad \left\| \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} - \mathbf{H} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \right\| \leq t$$

图 2-1 单应行矩阵计算公式

三、具体实施步骤

（一）设计模块

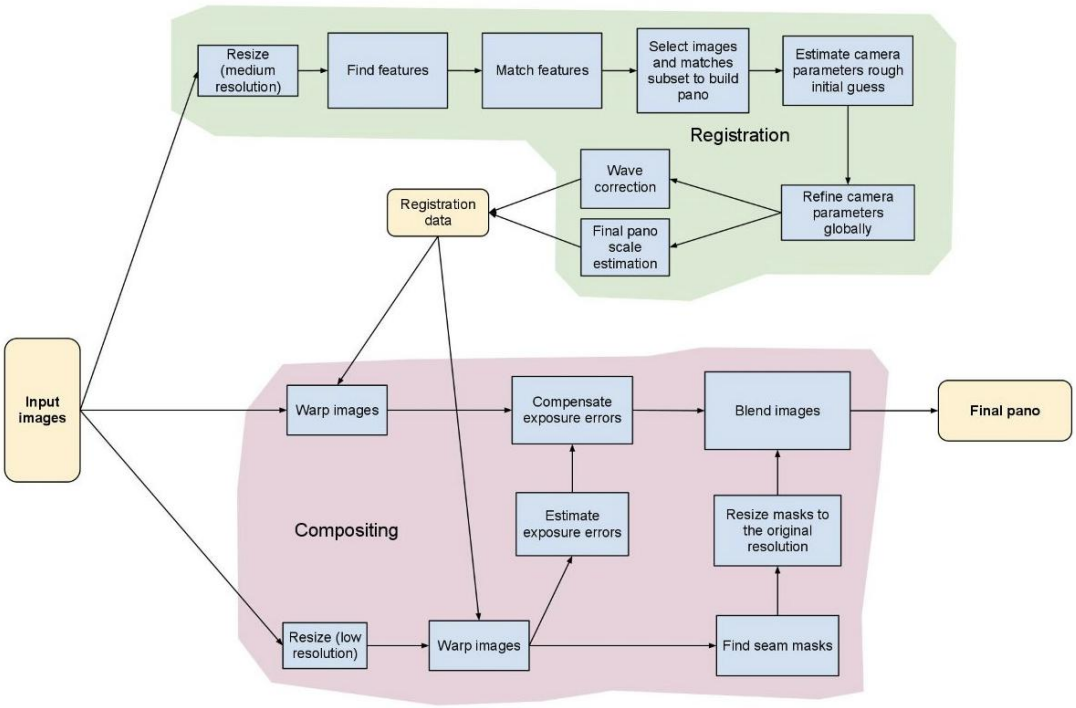

图 3-1 OpenCV 库 stitch 类实现模块

OpenCV 文档指出，stitch 类主要包含以下几个模块：功能查找和图像匹配，旋转估计，自动校准，图像变形，接缝估计，曝光补偿，图像对接。但我们具体实现时并不需要这么多的模块，考虑到实际情况，我将算法的具体实现分为：初始化模块，特征提取模块，特征匹配模块，翘曲变换模块，边缝融合模块，裁剪图像模块。同时，我们使用 OpenCV 库自带的库函数进行全景图像的拼接，将拼接后的图像传入裁剪图像模块，最后输出裁剪后的图像，进行学习研究。

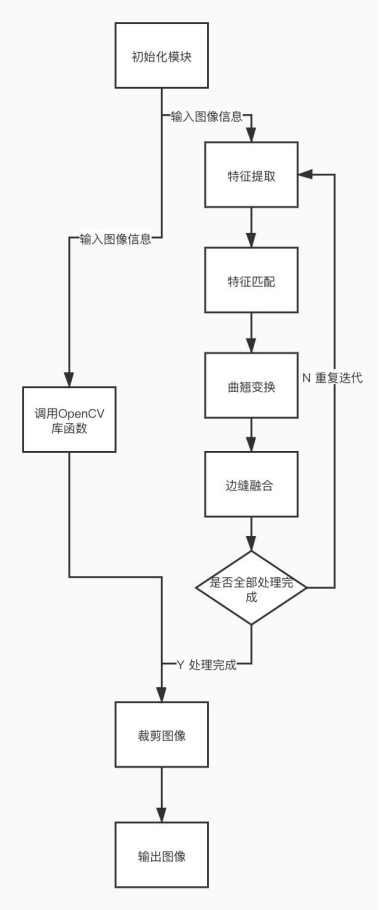

图 3-2 算法流程框架图

（二）初始化信息

该模块先后读入图片数量 img_num，图片名称 img_name 和图片格式 img_format，程序即可在 images 文件夹中定位到指定的需要拼接的图片。当读入图片信息之后，为防止图片过大导致拼接程序太慢，我们统一将图片缩放到宽为 500 的大小格式进行全景图拼接。

```
mstitcher = Stitcher()
#初始化，分别读入带拼接的照片名、数量和图片格式
img_dir = './images/'
img_num=int(input())
img_name=input()
img_format=input()
#形成图像列表
images =[]
for i in range(0,img_num):
    imagei = img_dir + str(img_name)+str(i+1)+"."+img_format
    imgi = cv2.imread(imagei)
    #裁剪图像
    r = 500.0/imgi.shape[1]
    dim = (500,int(imgi.shape[0]*r))
    imgi = cv2.resize(imgi, dim, interpolation=cv2.INTER_CUBIC)
    #cv2.imwrite(str(i)+'.png',imgi)
    images.append(imgi)
```

图 3-3 初始化部分代码

（三）特征提取

在该模块采用 SIFT 算法对指定图像进行特征点提取，将图像转变为灰度图像，使用 OpenCV 库中的 cv2.xfeatures2d.SIFT_create() 建立 SIFT 生成器，再使用函数 detectAndCompute()检测关键点并计算描述符，为了保证后续操作，将特征点集结果保存为 numpy 数组。

```python
def detectAndDescribe(self, img):
    # 将彩色图片转换成灰度图
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # 建立SIFT生成器
    img_dscrptr = cv2.xfeatures2d.SIFT_create()
    # 检测SIFT特征点，并计算描述子
    (kps, ftre) = img_dscrptr.detectAndCompute(gray, None)
    # 将结果转换成numpy数组
    kps = np.float32([kp.pt for kp in kps])
    # 返回特征点集，及对应的描述特征
    return (kps, ftre)
```

图 3-4 SIFT 特征提取部分代码

（四）特征匹配

对于将两张图片拼接成一张大图像，我们需要找到他们之间的重合点。利用这些重合点可以确定如何将第一张图像进行怎样的旋转缩放变形，使得能与下一张图像吻合拼接起来。这就是找到一种对应关系。在该模块需要使用 cv2.BFMatcher()建立暴力匹配器，使用 KNN 算法 knnMatch()检测来自 A 图和 B 图的 SIFT 特征点对，当最近距离与次近距离小于定义的 ratio 比率时，保留这个点对。

当筛选匹配对数超过 4 时，就可以计算视角变换矩阵。单应矩阵的作用是依据最佳匹配点，估计两幅图像中的相对方向变换。利用 cv2.findHomography()实现基于 RANSAC 的鲁棒算法可以计算出对应的单应矩阵，设置将点对视为内点的最大允许重投影错误阈值为 4。

```python
#建立匹配点
def matchKeypoints(self, kpsA, kpsB, ftreA, ftreB, ratio, reprojThresh):
    # 建立暴力匹配器
    matcher = cv2.BFMatcher()
    # 使用KNN检测来自A、B图的SIFT特征匹配对，K=2
    rawMatches = matcher.knnMatch(ftreB, ftreA, 2)
    matches = []
    for m in rawMatches:
        # 当最近距离跟次近距离的比值小于ratio值时，保留此匹配对
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            # 存储两个点在featuresA, featuresB中的索引值
            matches.append((m[0].trainIdx, m[0].queryIdx))
    # 当筛选后的匹配对大于4时，计算视角变换矩阵
    if len(matches) > 4:
        # 获取匹配对的点坐标
        ptsA = np.float32([kpsA[i] for (i, _) in matches])#左图找右图
        ptsB = np.float32([kpsB[i] for (_, i) in matches])#右找左
        # 计算视角变换矩阵
        #reprojThresh 为将点对视为内点的最大允许重投影错误阈值
        (H, status) = cv2.findHomography(ptsB, ptsA, cv2.RANSAC, reprojThresh)
        # 返回结果
        return (matches, H, status)
    return None
```

图 3-5 特征匹配模块代码

（五）翘曲变换

当我们得到单应矩阵之后，就相当于得到了位于左边图像的角度看位于右边图像的视角，所以，我们需要使用翘曲变换，将左图像通过单应矩阵变换到指定的图像尺寸，为后续图像拼接进行准备。其中，使用 cv2.warpPerspective()函数方法将左图像进行变换。

```
#翘曲变换
xh = np.linalg.inv(H)
f1 = np.dot(xh, np.array([0, 0, 1]))
f1 = f1 / f1[-1]
xh[0][-1] += abs(f1[0])
xh[1][-1] += abs(f1[1])  #
ds = np.dot(xh, np.array([imageA.shape[1], imageA.shape[0], 1]))
newy = abs(int(f1[1]))
newx = abs(int(f1[0]))
newsize = (int(ds[0]) + newx, int(ds[1]) + newy)
tmp = cv2.warpPerspective(imageA,xh,newsize)
```

图 3-6 翘曲变换部分代码

（六）边缝融合

将两张图片进行拼接的时候，两张图片的边缘会形成图像重叠的部分，可能会出现黑缝，鬼影等现象。为减少消除这种情况，采用像素逐渐过渡的方式去消弭这种肉眼上能观测出的突兀变化。即考虑加权融合的方法来让边缘的像素点为两张图像同坐标的像素点的加权和。alpha 值会随着图像离边缘的距离从 1.0 平滑地过度到 0.0。同时还需考虑到右图像边缘可能过度到 0.0 的情况，需要特判当 processwidth 等于 0 的特殊情况。

```
#边缝融合
def seamEstimation(self,tmp,leimg,riimg,newx,newy):
    alpha = 0.0
    processwidth = leimg.shape[1] - newx
    for x in range(newx,tmp.shape[1]):
        test_y = int((newy + leimg.shape[0])/2)
        if np.array_equal(tmp[test_y,x],np.array([0,0,0])):
            processwidth = x - newx
            break
    x_max = np.minimum(newx+riimg.shape[1],tmp.shape[1])
    y_max = np.minimum(newy+riimg.shape[0],tmp.shape[0])
    for x in range(newx,x_max):
        for y in range(newy,y_max):
            rx = x - newx
            ry = y - newy
            if not np.array_equal(tmp[y,x],np.array([0,0,0])) and not np.array_equal(riimg[ry,rx],np.array([0,0,0])):
                if processwidth==0:
                    alpha = np.minimum(1.0,2.0)
                else:
                    alpha = np.minimum(rx/processwidth,1.0)
                tmp[y,x] = alpha*riimg[ry,rx]+(1-alpha)*tmp[y,x]
            elif np.array_equal(tmp[y,x],np.array([0,0,0])):
                tmp[y,x] = riimg[ry,rx]
            elif np.array_equal(riimg[ry,rx],np.array([0,0,0])):
                tmp[y,x] = tmp[y,x]
    return tmp
```

图 3-7 边缝融合部分代码

当我们拼接完两张图像之后，将新图像迭代给 leftimage，将下一张图像传给 rightimage，

重复上述（三）到（六）模块，直到所有图像处理完成。

（七）裁剪图像

我们拼接完成图像都会在图片边缘产生不规则的黑边，为了去除这些黑边，需要采用裁剪图像。要将图像转换为灰度图像 gray，其次转换出二极值图像 thresh，使用孔洞填充方法将填充二极值图像，获取 thresh 二极值图矩形的最大轮廓，逐步缩小所选框图，直到框图不包含背景像素为止。

```python
#裁剪黑边
def Extra(self, img):
    img = cv2.copyMakeBorder(img, 10, 10, 10, 10, cv2.BORDER_CONSTANT, (0, 0, 0))
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)
    #孔洞填充
    thresh = self.fillHole(thresh)

    cnts, hierarchy = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnt = max(cnts, key=cv2.contourArea)

    mask = np.zeros(thresh.shape, dtype="uint8")
    x, y, w, h = cv2.boundingRect(cnt)
    cv2.rectangle(mask, (x, y), (x + w, y + h), 255, -1)
    minRect = mask.copy()
    sub = mask.copy()
    # 开始while循环，直到sub中不再有前景像素
    while cv2.countNonZero(sub) > 0:
        minRect = cv2.erode(minRect, None)
        sub = cv2.subtract(minRect, thresh)

    cnts, hierarchy = cv2.findContours(minRect.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnt = max(cnts, key=cv2.contourArea)
    x, y, w, h = cv2.boundingRect(cnt)

    # 使用边界框坐标提取最终的全景图
    img = img[y:y + h, x:x + w]
    return img
```

图 3-8 裁剪黑边部分代码

```python
#孔洞填充
def fillHole(self, img):
    result = img.copy()
    h,w = img.shape[:2]
    mask = np.zeros((h+2,w+2),np.uint8)
    cv2.floodFill(result, mask,(0,0),255)
    img2 = cv2.bitwise_not(result)
    imout = img | img2
    return imout
```

图 3-9 孔洞填充部分代码

四、成果展示

（一）A 图像：

OpenCV 拼接结果：

mstitcher 拼接结果：

（二）B 图像：

OpenCV 结果：

mstitcher 结果：

（三）C 图像：

OpenCV 拼接结果：

mstitcher 拼接结果：

（四）D 图像：

OpenCV 拼接结果：

mstitcher 拼接结果：

五、分析和总结

这次课程设计加深了我对 python 和 OpenCV 的理解。促进了我对全景图拼接的学习与了解。在这个项目之中，我深刻了解了 SIFT 算法和 RANSAC 算法的认识与理解。对全景图拼接的 OpenCV 官方库函数流程有了一定的理解与认识。在这个基础上，我根据自身的需要，简化设计了拼接函数的模块。

同时，在初步实现了拼接函数之后。根据不同的初始素材，出现了图像边缘有黑边的情况，为了解决这种情况，采用了翘曲变换与边缝融合的方法来解决。其次，拼接完成之后剪裁黑边的部分，会出现剪裁不完全的状况，为解决这种情况，采用了孔洞填充的方法。

但是，我的项目仍然有很多不足之处。第一，在焦距不同的情况下，我的代码无法将其完整拼接。究其原因是焦距不同导致翘曲变换之后的图像不规则，所以拼接无法进行下去。我们可以考虑将所有的图像进行圆柱畸变来解决这个问题。第二，对图像过大，过曝的拼接效果不好。因为代码中没有针对图像进行更详尽地归一化处理。第三，代码只能对图像进行从左至右的拼接，当输入图像没有按照指定顺序的时候，代码将出现错误。

总之，虽然完成了一份简单的全景图像拼接代码，但是仍然有很多的进步空间。这次课程设计加深了我对多媒体的认知。

六、附录

借鉴的博客：缩放图片：https://blog.csdn.net/sunmingyang1987/article/details/100387031
RANSAC 算法：https://blog.csdn.net/wwq0726/article/details/105068728
孔洞填充：https://blog.csdn.net/dugudaibo/article/details/84447196
全景图拼接一：https://andreame.com/2019/11/12/stitch.html
全景图拼接二：https://blog.csdn.net/qq_37374643/article/details/88930316
全景图拼接三：https://zhuanlan.zhihu.com/p/83225676