Exp3.计算机视觉实验报告

一、实验环境：Python2.7

二、分工

[redacted]：TODO1~4、TODO10、TODO11、TODO10（meshgrid、remap 优化版本整体设计，for 循环版本的插值设计）

[redacted]：TODO5~9、TODO12、TODO10（for 循环版本整体设计、优化版本的修正）

工作量比例大致 1:1

三、实验结果

（一）spherical warp

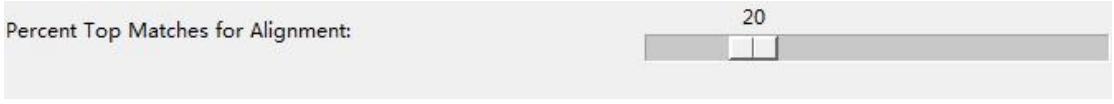
K1=-0.21

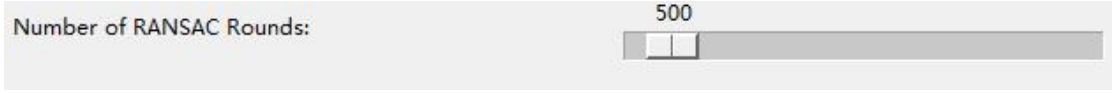
K2=0.26




（二）Alignment

1. 参数





RANSAC Threshold:	5.0
Focal Length (pixels):	678
k1:	-0.21
k2:	0.26

2. 结果图

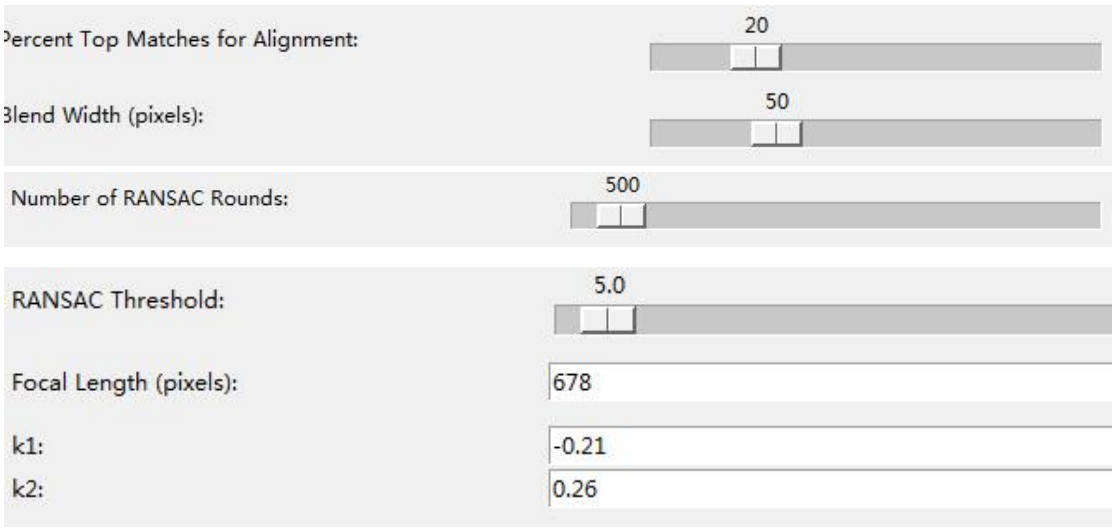
（1）translation

（2）homography

（三）Panorama

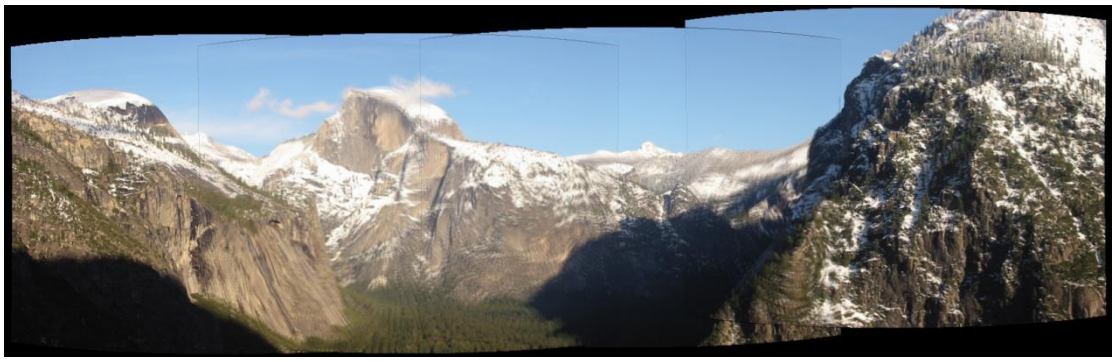
Yosemite

1. 参数



2. 结果图

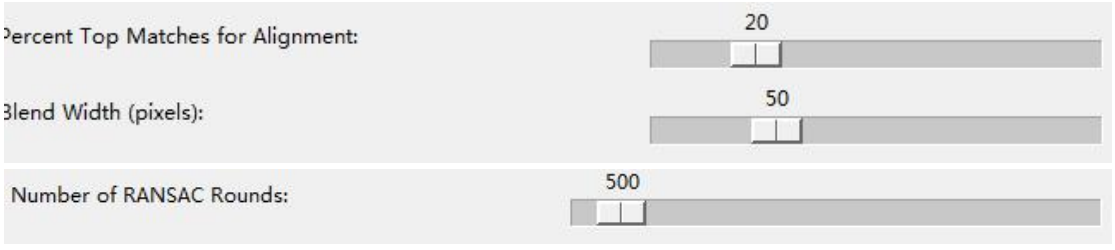
（1）Translation

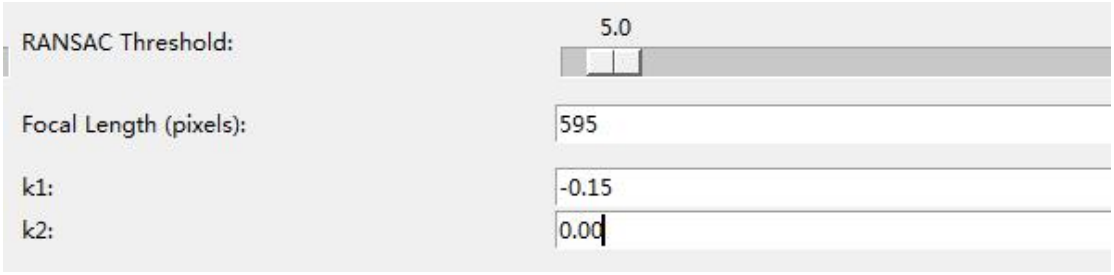


（2）homography

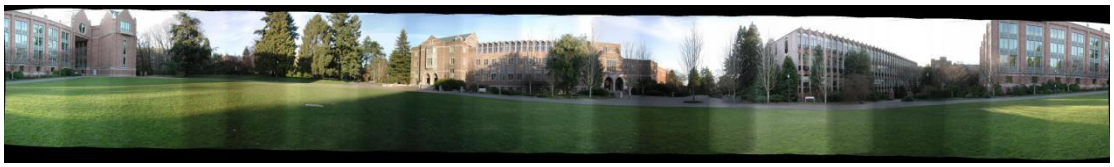
Campus

1. 参数





2. 结果图



四、 核心代码与优化

本次实验，在完成 alignment.py 的基础上，blend.py 中的 TODO10 是核心，该部分实现了每一张原图在累加区域 acc 中的重映射。按照实验说明，我们首先通过 for 循环的方式实现了代码，我们在运行程序时发现，程序运行极慢，其中在对 campus 进行 360 全景拼接时，运行时间为 20 分钟，因此我们用 meshgrid 和 remap 函数以及数组广播的技巧对 TODO10 进行了优化。

（一）for 循环版本

```
for i in range(x1, x2):
    for j in range(y1, y2):
        a = np.array([[i, j, 1]],dtype = float).T
        b = np.dot(np.linalg.inv(M), a)
        b /= b[-1]
        x = b[0]
        y = b[1]
#羽化
        if 0 <= x <= w and 0 <= y <= h:
            if x1 <= i <= x1 + blendWidth:
                weight = float(i - x1) / blendWidth

            elif x2 - blendWidth <= i <= x2:
                weight = float(x2 - i) / blendWidth

            else:
                weight = 1.0
#反卷绕的线性差值，其中 img1~4 是目标图像像素点逆映射到原图像位置时周围四个点的像
#素值
            img1 = img[int(math.floor(y)), int(math.floor(x))]
            img2 = img[int(math.floor(y)), int(math.ceil(x))]
            img3 = img[int(math.ceil(y)), int(math.floor(x))]
            img4 = img[int(math.ceil(y)), int(math.ceil(x))]
```

```
                img5 = (math.ceil(y) - y) * img1 + (y - math.floor(y)) * img3
                img6 = (math.ceil(y) - y) * img2 + (y - math.floor(y)) * img4

                img7 = (math.ceil(x) - x) * img5 + (x - math.floor(x)) * img6

#原图中黑色像素的权值置零
                if (img7 - [0, 0, 0]).any() == False:
                    weight = 0.0

                acc[j, i, 0:3] += img7* weight

                acc[j, i, 3] += weight
'''
return acc
```

代码中各部分说明已有注释：主要有羽化、反卷绕的差值和黑色像素处理这三个部分

（二）优化版本

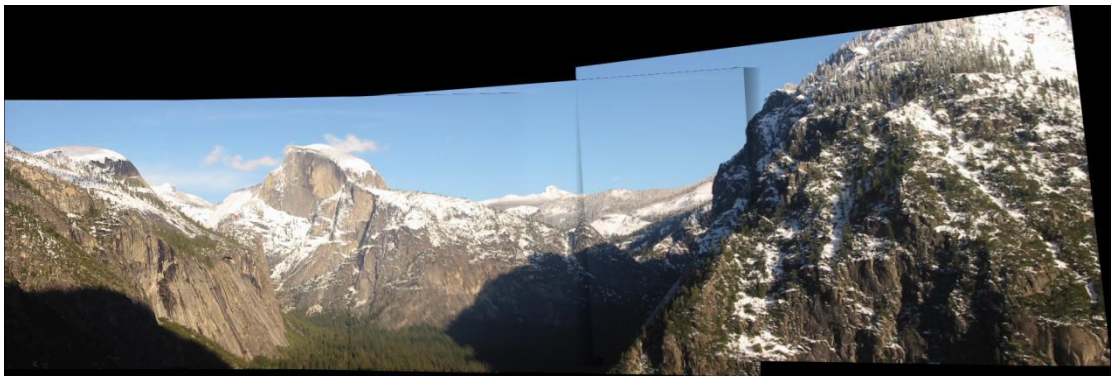
```
X1, y1, x2, y2 = imageBoundingBox(img, M)
#创建目标图像的坐标矩阵
row_num = y2-y1
col_num = x2-x1
x_range = np.arange(x1, x2)
y_range = np.arange(y1, y2)
(x_mesh, y_mesh) = np.meshgrid(x_range, y_range)
#重塑坐标矩阵，行数为 3（x，y，z），列数为 row_num*col_num，以实现目标图所有像素
#的逆映射
one = np.ones((row_num, col_num))
allLoc = np.dstack((x_mesh, y_mesh, one))
allLoc = allLoc.reshape((row_num*col_num, 3))
allLoc = allLoc.T
 # 卷绕到了原来的坐标
loc2 = np.dot(np.linalg.inv(M), allLoc)
loc2 = loc2 / loc2[2]   # z 设为 1
#用 remap 函数实现重映射，即找到目标图中所有像素点在原图中的位置，并将对应像素的
#像素值进行传送
map_x = loc2[0].reshape((row_num, col_num)).astype(np.float32)   # 挑出来 做 个类型转换
map_y = loc2[1].reshape((row_num, col_num)).astype(np.float32)
#生成目标图
dst = cv2.remap(img, map_x, map_y, cv2.INTER_LINEAR)
#权重矩阵
weight = np.ones((row_num, col_num))
```

```
#羽化帽子向量
blendc = np.arange(0+1.0/blendWidth, 1+1.0/blendWidth, 1.0/blendWidth)
#对权重矩阵进行羽化
weight[:, 0:blendWidth] = blendc
weight[:, col_num-blendWidth:col_num] = -np.sort(-blendc)
#图像暂存矩阵
acc1 = np.zeros((row_num, col_num,3))
acc1 += np.multiply(dst, weight.reshape(row_num, col_num, 1))
#acc[y1:y2, x1:x2, 0:3] += np.multiply(dst, weight.reshape(row_num, col_num, 1))
#处理原图黑色像素值
weightc = acc1.sum(axis=2)
weightc = (weightc > 0)*1.0
weight = np.multiply(weight, weightc)
#对 acc 进行修改
acc[y1:y2, x1:x2, 0:3] += acc1
acc[y1:y2, x1:x2, 3] += weight
```

优化代码的说明已写入注释，此代码用矩阵运算代替了 for 循环，因此极大提高了程序运行效率。

五、 代码修正

在对 TODO10 优化版本进行调试时，我们发现在图像拼接处存在边框的阴影，如下图：



我们初步认为这是由于黑色像素的权重置零错误导致的

我们在出版代码中找到了错误原由（改代码为错误代码，已注释）：

#acc[y1:y2, x1:x2, 0:3] += np.multiply(dst, weight.reshape(row_num, col_num, 1))

经分析，我们发现：由于每一张图片经过映射后，直接在 acc 矩阵中对应的位置对黑色像素进行判断，而除了第一张图，每一张图都会受到前一张图 blendwith 区域像素的影响，进而导致本应判定为黑色像素的点判定失败。

因此，我们的解决方案是：用 acc1 暂存重映射图像，并在 acc1 中进行黑色像素的判定，然后再将 acc1 赋值给 acc 的相应位置。

修改后效果图如下：

六、 体会

这次实验，除了完成了对全景图的实践体会，更重要的是体会到了，发现问题、分析问题并解决问题的艰辛以及喜悦。