

图像处理基础

（实验2）



学 院（系）： 电子信息与电气工程学部

专 业： 计算机科学与技术

班 级： 电计1704

学 生 姓 名： 谢玉宁

学 号： 201792260

完 成 日 期： 2020年5月2日

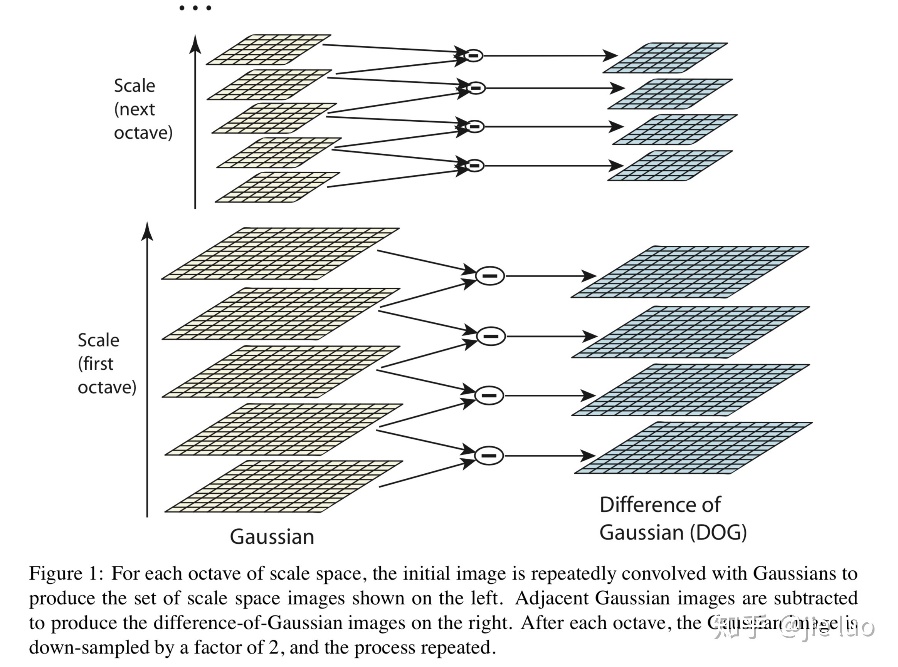
**实验目标：**用SIFT实现图像全景拼接

**实现原理:**

1. 实现SIFT（Scale-invariant feature transform）：
2. 构建高斯金字塔：

本实验取S=6，n= S-3 = 3

构建6倍频程高斯金字塔、每层内部不同尺度图片相减，得到差分金字塔DOG

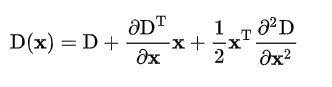


图表 1 高斯金字塔与DOG（S=5）

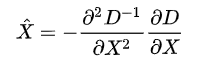
1. 获得极值点：

对比DOG像素点相邻的26个邻域，选极大值或极小值作候选keypoint，对这些关键点在进行调整，以实现精确定位。方法如下：

对尺度空间DOG函数进行曲线拟合。利用DOG函数在尺度空间的泰勒展开式为：



求导并让方程等于零，可以得到极值点的偏移量为：

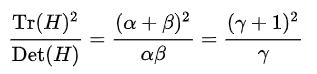


通过对x求偏导并将结果置为零，我们可以简单地计算出方程的极值，从而得到子像素特征点的位置，这些子像素点可以增加匹配和算法稳定性的机率。

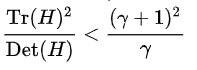
若子像素点与近似特征点间的偏移量大于0.5，则按照偏移近似特征点的方向相应改变（移动）特征点，然后再把该点当做近似特征点，重复该操作（(Lowe算法里最多迭代5次），直到子像素点与近似特征点间的偏移量小于等于0.5。

得到最终候选点的精确位置与尺度X  ，将其代入公式求得D(X)  ，求其绝对值。如果其绝对值低于阈值的将被删除。

由于边缘的局部特征不明显，所以还需要删除关键点的边缘效应。



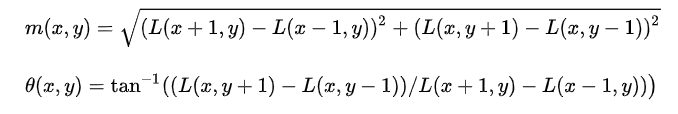
只要满足



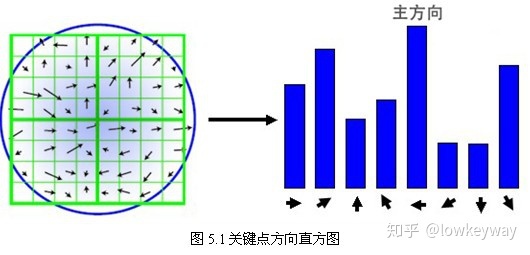
则保留关键点，否则删除。

1. 获得关键点的主方向

采集其所在高斯金字塔图像3σ邻域窗口内像素的梯度和方向分布特征。梯度的模值和方向



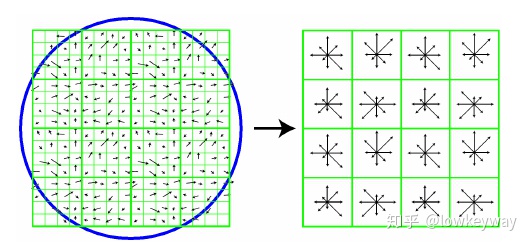
按尺度采样的3σ原则，邻域窗口半径为 3 \* 1.5σ\_oct



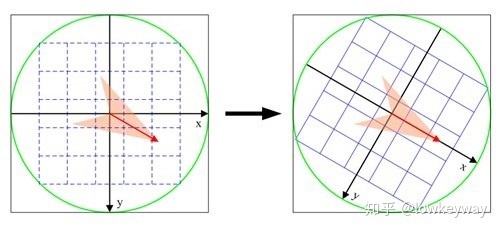
直方图的峰值则代表了该关键点处邻域梯度的主方向，即作为该关键点的方向；其他的达到最大值80%的方向可作为辅助方向。至此，将检测出的含有位置、尺度和方向的关键点即是该图像的SIFT特征点。

1. 生成关键点描述子

对于每一个关键点，拥有三个信息：位置、尺度以及方向。接下来就是为每个关键点建立一个描述符，用一组向量将这个关键点描述出来，使其不随各种变化而改变，比如光照变化、视角变化等等。这个描述子不但包括关键点，也包含关键点周围对其有贡献的像素点，并且描述符应该有较高的独特性，以便于提高特征点正确匹配的概率。SIFT描述子是关键点邻域高斯图像梯度统计结果的一种表示。通过对关键点周围图像区域分块，计算块内梯度直方图，生成具有独特性的向量，这个向量是该区域图像信息的一种抽象，具有唯一性。



每一个小格都代表了特征点邻域所在的尺度空间的一个像素 ，箭头方向代表了像素梯度方向，箭头长度代表该像素的幅值。然后在4×4的窗口内计算8个方向的梯度方向直方图。绘制每个梯度方向的累加可形成一个种子点。



将邻域内的采样点分配到对应的子区域内，将子区域内的梯度值分配到8个方向上，计算其权值, 插值计算每个种子点八个方向的梯度.

如上统计的4\*4\*8=128个梯度信息即为该关键点的特征向量。特征向量形成后，为了去除光照变化的影响，需要对它们进行归一化处理，对于图像灰度值整体漂移，图像各点的梯度是邻域像素相减得到，所以也能去除。

**任务完成情况：**

1. 所有函数除了最后stitch类均为手写，运用了之前的代码插值，改变图像尺寸，卷积等（除了读写图像，数组分片，numpy自带的数组级操作之外）；
2. 完成了对灰度图像构建高斯金字塔，差分金字塔；
3. 完成了特殊点的计算
4. 完成了主方向计算
5. 完成了特征点的计算
6. 根据视频自带的代码修改，实现图像关键点连线，图像缝合。

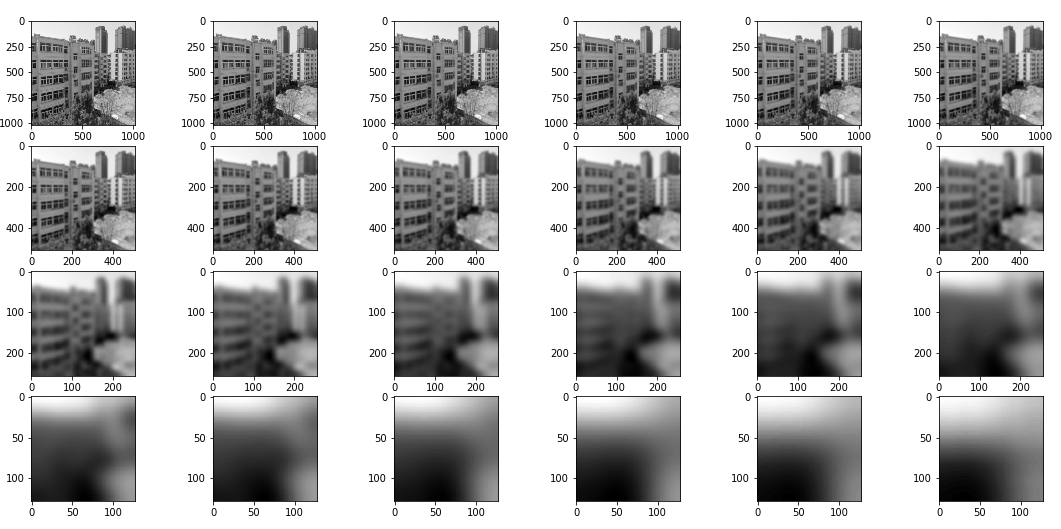
**结果展示：**

1. 图像拼接结果：

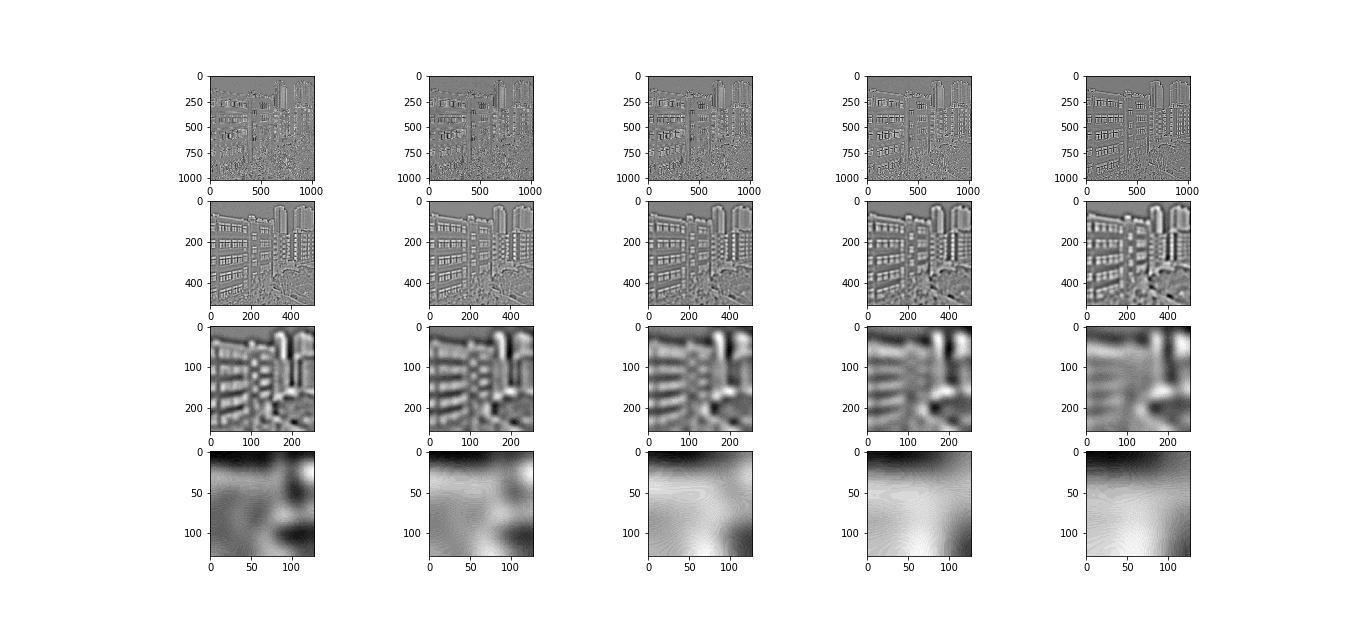
输入图像信息：1024\*1024像素

构建高斯金字塔和差分金字塔：



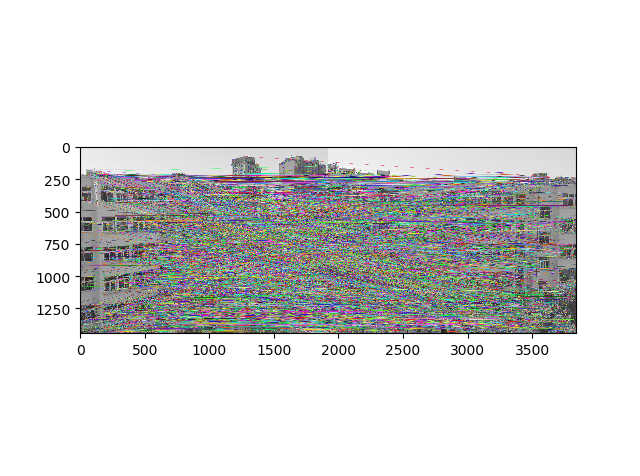
图表 2 Gaussian Pyramid（P1）



图表 3 DOG（P1）



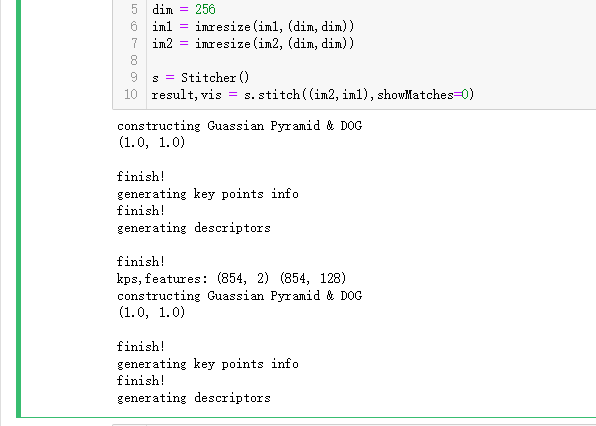
图表 4 关键点



图表 5 关键点匹配连线



图表 6 拼接结果图



图表 7 运行截图

**收获与讨论：**

这是我完全复现一个经典算法的为数不多的经历之一，通过这次实验我也掌握了一些查阅资料的能力。同时我感到，在深度学习流行的今天，传统的经典算法在一段时间内还是不会被替代的，SIFT算法算是图像处理中最基本的算法，效果也非常的好，做完成就感也很大，但是我算法复杂度过高，用opencv封装的SIFT算法，几秒钟能完成的工作，我的算法需要几分钟，可见优化算法的重要性!

本次试验代码及函数均为手写，应用了之前作业的函数，如插值改变图像尺寸，旋转，卷积操作等，并在此基础上实现对了SIFT算法以及图像缝合（全景拼接），最终成功完成实验。

本试验让我对图像处理又有了更深刻的认识，更加熟悉图像在计算机中的储存结构以及其中的细节，灰度图像与RGB图像之间的关系以及转化；对图像进行不同操作的效果（滤波采样等），加深了我对SIFT算法的理解，

通过本次实验，再次让我认识到了编程中的一些技巧，比如函数的重用，分函数写可能对某个简单的任务来说显得麻烦了一些，但是对于较大代码量的工程来说，不仅方便日后的重用，也方便了调试。同时学到了一些调试的小技巧，首先测试样例不能太大，这样会浪费时间，其次要先分别对不同函数进行调试，这样容易快速定位bug，最后将函数整合起来。

本实验工作量比较大，一开始走了错误的路线，面向过程编程，卡在一些小bug上比如float32和np.float32竟然不是一个类型（储存方式一样，不过np.float32可能被np封装了多了一些属性，但是opencv直接判定导致），结果花了很长时间。

最后感谢马老师和王老师的讲解，疫情期间虽然没见到真人，但是还是学到了该学的东西。

**代码：**

# -\*- coding: utf-8 -\*-

"""

Created on Sun Apr 12 10:24:38 2020

@author: Lenovo

"""

from scipy.interpolate import griddata

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as pli

import pickle

from PIL import Image

import cv2

def Xrot(X, angle):

anglePi = angle \* np.pi / 180.0

cos = np.cos(anglePi)

sin = np.sin(anglePi)

Rot\_M = np.array([[cos,-sin],

[sin, cos]])

Y = np.matmul(Rot\_M,X)

return Y

def Irot(Image,Angle):

h = Image.shape[0]

w = Image.shape[1]

x = np.linspace(h/2-0.5,-h/2+0.5,h)

y = np.linspace(-w/2+0.5,w/2-0.5,w)

xcoord,ycoord = np.meshgrid(y,x)

X = np.concatenate((xcoord.flatten()[np.newaxis,:],ycoord.flatten()[np.newaxis,:]),axis = 0)

lvector = Image.reshape(Image.shape[0]\*Image.shape[1])

Y = Xrot(X,Angle)

scale\_x = int(np.floor(Y[0].min())),int(np.ceil(Y[0].max()))

scale\_y = int(np.floor(Y[1].min())),int(np.ceil(Y[1].max()))

x\_range,y\_range = scale\_x[1]-scale\_x[0],scale\_y[1]-scale\_y[0]

x = np.linspace(x\_range/2-0.5,-x\_range/2+0.5,x\_range)

y = np.linspace(-y\_range/2+0.5,y\_range/2-0.5,y\_range)

new\_xcoord,new\_ycoord = np.meshgrid(-x,-y)

grid\_z = griddata(Y.T, lvector, (new\_xcoord,new\_ycoord), method='linear')

#grid\_z = mlab.griddata(new\_xcoord.flatten(),new\_ycoord.flatten(),lvector,)

plt.imshow(grid\_z)

return grid\_z

def Convolve(I,F):

iw,ih = I.shape

fw,fh = F.shape

Conv\_I = np.zeros\_like(I)

Image\_pad = np.pad(I,(((fh-1)//2,(fh-1)//2),((fw-1)//2,(fw-1)//2)), 'reflect')

func = lambda x,y:np.sum(Image\_pad[x:x+fh,y:y+fw]\*F)

for i in range(ih):

for j in range(iw):

Conv\_I[i,j] = func(i,j)

return Conv\_I

def downsample(I,step = 2):

return I[::step,::step]

def RGB2Gray(image):

row,col = image.shape[:2]

gray = np.zeros((row,col),dtype = 'float')

gray = 0.11 \*image[:,:,0].squeeze() +0.59\*image[:,:,1].squeeze() +0.3\*image[:,:,2].squeeze()

return gray.astype('uint8')

def imresize(original\_image,target\_size):

i = Image.fromarray(original\_image)

ii = i.resize(target\_size[::-1],Image.BILINEAR)

image = np.asarray(ii)

return image

def show(I,gray = 0):

I = I.copy()

if I.ndim==2:

plt.imshow(I,cmap='gray')

elif gray:

I = RGB2Gray(I)

plt.imshow(I,cmap = 'gray')

else:

plt.imshow(I)

def display(GP,optave\_num=3,scale\_num=6):

counter = 1

for i in range(optave\_num):

for j in range(scale\_num):

plt.subplot(optave\_num,scale\_num,counter) #要生成两行两列，这是第一个图plt.subplot('行','列','编号')

plt.imshow(GP[i][j],cmap='gray')

counter+=1

def spot(im3,kps,mds):

for i,j in kps:

if mds[kps.index([i,j])]==0 or mds[kps.index([i,j])]==45:

im3[int(i),int(j)]= [255,0,0]

elif mds[kps.index([i,j])]==180 or mds[kps.index([i,j])] == 225:

im3[int(i),int(j)]= [0,255,0]

elif mds[kps.index([i,j])]==90 or mds[kps.index([i,j])] == 135:

im3[int(i),int(j)]= [0,0,255]

else:

im3[int(i),int(j)]= [255,255,255]

show(im3)

def load\_pickle(GP\_file,DOG\_file=None):

path = r'C:\Users\Lenovo\Desktop\pickle\\'

with open(path+GP\_file+'.pkl','rb') as f:

GP =pickle.load(f)

if DOG\_file == None:

return GP

with open(path+DOG\_file+'.pkl','rb') as f:

DOG =pickle.load(f)

return GP,DOG

def preprocess(img):

if img.ndim==3:

img = RGB2Gray(img)

target\_size = (2\*\*int(np.log2(min(img.shape))),2\*\*int(np.log2(min(img.shape))))

i = imresize(img,target\_size).copy()

ratio\_x = img.shape[0]/i.shape[0]

ratio\_y = img.shape[1]/i.shape[1]

return i,(ratio\_x,ratio\_y)

def Guassian\_Kernel(sigma,dim):

temp = [t - (dim//2) for t in range(dim)]

assistant = []

for i in range(dim):

assistant.append(temp)

assistant = np.array(assistant)

temp = 2\*sigma\*sigma

kernel = (1.0/(temp\*np.pi))\*np.exp(-(assistant\*\*2+(assistant.T)\*\*2)/temp)

return kernel

#生成一层的高斯图像

#sigma0是该octave的第一层sigma

def GP\_Octave(I,Scale\_num,k,sigma0):

octave = []

# sigma = [k\*\*i for i in range(Scale\_num)]\*sigma0

# kernel\_dim = [int(6\*i+2) if (6\*i)%2 else int(6\*i+1) for i in sigma]

m = min(I.shape)

for i in range(Scale\_num):

sigma = k\*\*i \* sigma0

kernel\_dim = int(6\*sigma+2) if int(6\*sigma)%2 else int(6\*sigma+1)

if kernel\_dim > m:

kernel\_dim = int(m+2) if int(m)%2 else int(m)+1

K = Guassian\_Kernel(sigma,kernel\_dim)

octave.append(Convolve(I,K))

return octave

def generate\_DOG(I,Scale\_num = 6,sigma0=1.5,Octave\_num=None):

print('constructing Guassian Pyramid & DOG')

I,ratio = preprocess(I)

print(ratio)

if(I.ndim==3):

Image = RGB2Gray(I)

else:

Image = I.copy()

n = Scale\_num-3

if Octave\_num is None:

Octave\_num = int(np.log2(min(Image.shape[0],Image.shape[1]))) - 3

k = 2\*\*(1./n)

G\_Pyramid = []

init\_sigma = [sigma0\*(2\*\*i) for i in range(Octave\_num)]

# kernel\_dim = [int(6\*i+2) if (6\*i)%2 else int(6\*i+1) for i in l]

for i in range(Octave\_num):

G\_Pyramid.append(GP\_Octave(Image,Scale\_num,k,init\_sigma[i]))

Image = downsample(Image,step = 2)

DOG = [[G\_Pyramid[i][j+1].astype(int) - G\_Pyramid[i][j].astype(int) for j in range(len(G\_Pyramid[0])-1)] for i in range(len(G\_Pyramid))]

print('\nfinish!')

return G\_Pyramid,DOG

#p,D = generate\_DOG(im1,6,sigma0=1.5,Octave\_num=5)

#with open(r'C:\Users\Lenovo\Desktop\pickle\GP.pkl','wb') as f:

# pickle.dump(p,f)

#

#with open(r'C:\Users\Lenovo\Desktop\pickle\DOG.pkl','wb') as f:

# pickle.dump(D,f)

def Adjust(DOG,o,s,x,y):

n = 3

img\_border = 5

I = DOG[o][s].astype(float)

for i in range(5):

if s < 1 or s > n or y < img\_border or y >= I.shape[1] - img\_border or x < img\_border or x >= I.shape[0] - img\_border:

return [None]\*4

I\_prev = DOG[o][s-1].astype(np.float32)

I = DOG[o][int(s)].astype(np.float32)

I\_next = DOG[o][int(s+1)].astype(np.float32)

dD = np.array([I[x,y+1] - I[x,y-1],

I[x+1,y] - I[x-1,y],

I\_next[x,y] - I\_prev[x,y]],dtype=np.float32)\*0.5

v2 = I[x,y] \* 2

Dxx = (I[x,y+1] + I[x,y-1] - v2)

Dyy = (I[x+1,y] + I[x-1,y] - v2)

Dss = (I\_next[x,y] + I\_prev[x,y] - v2)

Dxy = (I[x+1,y+1] - I[x+1,y-1] - I[x-1,y+1] + I[x-1,y-1]) \* 0.25

Dxs = (I\_next[x,y+1] - I\_next[x,y-1] - I\_prev[x,y+1] + I\_prev[x,y-1]) \* 0.25

Dys = (I\_next[x+1,y] - I\_next[x-1,y] - I\_prev[x+1,y] + I\_prev[x-1,y]) \* 0.25

H=np.array([[Dxx, Dxy, Dxs],

[Dxy, Dyy, Dys],

[Dxs, Dys, Dss]],dtype=np.float32)

X = -np.matmul(np.linalg.pinv(H),dD)

dx,dy,ds = X

if (np.abs(X) < 0.5).all():

break

x+=int(round(dx))

y+=int(round(dy))

s+=int(round(ds))

#迭代 5 次都没找到，丢弃

else:

return [None]\*4

#判断找到的点是否在边界内，边界外舍去

if s < 1 or s > n or y < img\_border or y >= I.shape[1] - img\_border or x < img\_border or x >= \

I.shape[0] - img\_border:

return [None]\*4

#判断是可能是噪声，可能是噪声舍去

dg = dD.dot(np.array([dx, dy, ds]))

respone = I[x,y] + dg \* 0.5

if np.abs(respone) \* n < 0.04 \*255:

return [None]\*4

# 利用Hessian矩阵的迹和行列式计算主曲率的比值

Tr = Dxx + Dyy

det = Dxx \* Dyy - Dxy \* Dxy

if det<=0 or Tr \* Tr / det >= 12.1:

return [None]\*4

Key\_point = []

Key\_point.append(int((x + dx) \* 2\*\*o))

Key\_point.append(int((y + dy) \* 2\*\*o))

return x,y,s,Key\_point

def MainPoint(GP\_layer,o,s,r,c):

sigma\_oct = 1.52\*2\*\*(s/3+o)

radius = int(np.ceil(3\* 1.5\*sigma\_oct))

dim = 2\*radius+1

K = Guassian\_Kernel(1.5\*sigma\_oct,dim)

dx = np.zeros((2\*radius+1,2\*radius+1),dtype=np.float32)

dy = np.zeros((2\*radius+1,2\*radius+1),dtype=np.float32)

sita = np.full\_like(dx,-4.)

for i in range(-radius,radius+1):

x = r+i

if x<1 or x>GP\_layer.shape[0]-2:

continue

for j in range(-radius,radius+1):

y = c+j

if y<1 or y>GP\_layer.shape[0]-2:

continue

dx[i+radius,j+radius] = GP\_layer[x+1,y].astype(np.float32) - GP\_layer[x-1,y].astype(np.float32)

dy[i+radius,j+radius] = GP\_layer[x,y+1].astype(np.float32) - GP\_layer[x,y-1].astype(np.float32)

sita[i+radius,j+radius] = np.arctan2(dy[i+radius,j+radius],dx[i+radius,j+radius])

grad = (dx\*\*2 + dy\*\*2)\*\*0.5 \* K

sita = sita\*180/np.pi +180

#统计

hist = np.array([0.]\*8)

for i in range(2\*radius+1):

for j in range(2\*radius+1):

if sita[i,j]<0:

continue

hist[int((sita[i,j])//45.1)] +=grad[i,j]

main\_dir = np.argmax(hist)\*45

return main\_dir

def Key\_points\_info(GP,DOG):

print('generating key points info')

kps=[]

mds=[]

kps\_info = []

O = len(DOG)

S = len(DOG[0])

boarder = 5

for o in range(O):

for s in range(1,S-1):

I\_pre, I\_cur,I\_next = DOG[o][s-1],DOG[o][s],DOG[o][s+1]

stride = max(1,min(8,2\*\*(int(np.log2(min(I\_cur.shape)))-6)))

stride = 1

for i in range(boarder,I\_cur.shape[0]-boarder,stride):

for j in range(boarder,I\_cur.shape[1]-boarder,stride):

val = I\_cur[i,j]

eight\_neiborhood\_prev = I\_pre[i-1:i+2,j-1:j+2]

eight\_neiborhood = I\_cur[i-1:i+2,j-1:j+2]

eight\_neiborhood\_next = I\_next[i-1:i+2,j-1:j+2]

if ((val > 0 and (val >= eight\_neiborhood\_prev).all() and (val >= eight\_neiborhood).all() and (val >= eight\_neiborhood\_next).all())

or (val < 0 and (val <= eight\_neiborhood\_prev).all() and (val <= eight\_neiborhood).all() and (val <= eight\_neiborhood\_next).all())):

x,y,scale,kp = Adjust(DOG,o,s,i,j)

if kp == None:

continue

md = MainPoint(GP[o][scale],o,s,x,y)

kps.append(kp)

mds.append(md)

kps\_info.append([o,scale,x,y,md,kp])

print('finish!')

return kps\_info

def calcSIFTDescriptor(img,x,y,direction,radius,d=4,n=8):

radius = 8

radius = int(radius)

features = []

# cos\_t = np.cos(direction \* (np.pi / 180)) # 余弦值

# sin\_t = np.sin(direction \* (np.pi / 180)) # 正弦值

rang = 2\*int(np.ceil(1.415\*radius)+1)

I = img[x-rang:x+rang,y-rang:y+rang]

if I.shape[0]!=I.shape[1] or I.shape[0]==0 or I.shape[1]!=2\*rang:

return None

rot\_I = Irot(I,-direction)

x,y = rot\_I.shape[0]//2,rot\_I.shape[1]//2

for r in range(-2,2):

for c in range(-2,2):

hist = [0.]\*8

for i in range(x+r\*radius,x+r\*radius+radius):

for j in range(y+c\*radius,y+c\*radius+radius):

dx = rot\_I[i+1,j].astype(np.float32) - rot\_I[i-1,j].astype(np.float32)

dy = rot\_I[i,j+1].astype(np.float32) - rot\_I[i,j-1].astype(np.float32)

theta = np.arctan2(dy,dx)\*180/np.pi+180

grad = (dx\*\*2 + dy\*\*2)\*\*0.5

hist[int((theta)//45.1)] +=grad

features.append(hist)

features = np.array(features,dtype='float32').flatten()

nrm2 = 0

length = d \* d \* n

for k in range(length):

nrm2 += features[k] \* features[k]

thr = np.sqrt(nrm2) \* 0.2

nrm2 = 0

for i in range(length):

val = min(features[i], thr)

features[i] = val

nrm2 += val \* val

nrm2 = 512 / max(np.sqrt(nrm2), 1.19209290E-07)

for k in range(length):

features[k] = min(max(features[k] \* nrm2,0),255)

return features

def kps\_feature(pic\_name):

kps\_info = []

feature = []

with open(r'C:\Users\Lenovo\Desktop\pickle\\'+pic\_name+'\_kps\_info.pkl','rb') as f:

kps\_info = pickle.load(f)

with open(r'C:\Users\Lenovo\Desktop\pickle\\'+pic\_name+'\_descriptors.pkl','rb') as f:

feature = pickle.load(f)

kps = [kp[5] for kp in kps\_info]

return kps,feature

class Stitcher:

def stitch(self,images,ratio=0.75,reprojThresh=4.0,showMatches=False):

(imageB,imageA) = images

# (kpsA,featuresA) = self.detectAndDescribe2(imageA)

# (kpsB,featuresB) = self.detectAndDescribe2(imageB)

kpsA,featuresA = kps\_feature('building2')

kpsB,featuresB = kps\_feature('building1')

print('matching and stitching')

M = self.matchKeypoints(kpsA,kpsB,featuresA,featuresB,ratio,reprojThresh)

if M is None:

return None

(matches,H,status,rawMatches) = M

result = cv2.warpPerspective(imageA,H,(imageA.shape[1]+imageB.shape[1],imageA.shape[0]))

# self.cv\_show('result',result)

result[0:imageB.shape[0],0:imageB.shape[1]] = imageB

# self.cv\_show('result',result)

vis = None

if showMatches:

vis = self.drawMatches(imageA,imageB,kpsA,kpsB,rawMatches,status)

print('finish!')

print(result,vis)

return result,vis

def drawMatches(self,img1,img2,kp1,kp2,matches,status):

# matches = sorted(matches,key = lambda x:x.distance)

good=[]

for m,n in matches:

if m.distance <0.75\*n.distance:

good.append([n])

img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=2)

plt.show(im3)

return im3

def detectAndDescribe(self,image):

gray = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

descriptor = cv2.xfeatures2d.SIFT\_create()

kps,features = descriptor.detectAndCompute(image,None)

kps =np.float32([kp.pt for kp in kps])

return kps,features

def detectAndDescribe2(self,image):

GP,DOG = generate\_DOG(image)

# GP,DOG = load\_pickle('building1\_GP','building1\_DOG')

# display(DOG,3,5)

kps\_info = Key\_points\_info(GP,DOG)

# kps\_info = load\_pickle('building1\_kps\_info')

location = [kp[0:4] for kp in kps\_info]

mds = [kp[4] for kp in kps\_info]

kps = [kp[5] for kp in kps\_info]

print('generating descriptors')

kps\_reduce=[]

descriptors = []

for i in range(len(kps\_info)):

o,s,x,y = location[i]

md = mds[i]

sigma\_oct = 1.52\*2\*\*(s/3+o)

radius = int(np.ceil(3\* 1.5\*sigma\_oct))

f = calcSIFTDescriptor(GP[o][s],x,y,md,radius)

if f is None:

continue

kps\_reduce.append(kps[i])

descriptors.append(f)

print('\nfinish!')

print('kps,features:',np.array(kps\_reduce).shape,np.array(descriptors,dtype='float32').shape )

return np.array(kps\_reduce,dtype=np.float32),np.array(descriptors,dtype='float32')

def matchKeypoints(self,kpsA,kpsB,featuresA,featuresB,ratio,reprojThresh):

matcher = cv2.BFMatcher()

rawMatches = matcher.knnMatch(featuresA,featuresB,2)

# matches = matcher.match(featuresA,featuresB)

# matches = sorted(matches, key=lambda x: x.distance)

matches = []

for m in rawMatches:

if len(m) == 2 and m[0].distance < m[1].distance\*ratio:

matches.append((m[0].trainIdx,m[0].queryIdx))

if len(matches)>4:

ptsA = np.float32([kpsA[i] for (\_,i) in matches])

ptsB = np.float32([kpsB[i] for (i,\_) in matches])

H,status = cv2.findHomography(ptsA,ptsB,cv2.RANSAC,reprojThresh)

return matches,H,status,rawMatches

def cv\_show(name,result):

cv2.imshow(name,result)

im1\_path = r'C:\Users\Lenovo\Desktop\kb1.jpg'

im1 = pli.imread(im1\_path)

im2\_path = r'C:\Users\Lenovo\Desktop\kb2.jpg'

im2 = pli.imread(im2\_path)

dim = 1024

im1 = imresize(im1,(dim,dim))

im2 = imresize(im2,(dim,dim))

s = Stitcher()

result,vis = s.stitch((im1,im2),showMatches=0)