CV HW2 报告

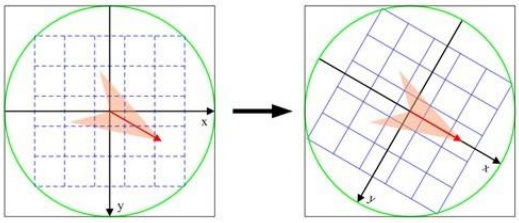
519030910383 张哲昊

detect_blobs：首先使用不同σ(σ = $\sigma_0 * s^k$)的高斯滤波(卷积核大小使用 cv2 对于σ的默认大小)，对原灰度图像进行高斯滤波。（σ_0=1，s = 1.5，σ空间的维度设置为 10 可以处理除了 graf1 和 graf2 以及 wall1 和 wall3 的其他所有图片）。之后利用 difference of gaussion (Dog)来近似 NLoG 获得 sigma 空间的的图像，存储为三维 numpy 矩阵 dog。接下来将检测 dog 中 sigma 空间和 xy 空间的极值来定位 blob。

图 1 bikes1 的 blob 样例

使用 3 个 for 循环来遍历每一个 sigma，h，w 来寻找三个方向的极值（导致 if 的条件非常冗长，可以通过 scipy.ndimage.filters maximum_filter 来简化）。同时为了剔除对比度较小的点，剔除了 dog 图像中的值小于某个阈值的 blob（该阈值设置为）。可以通过上图发现，图像中的关键点及其代码均被很好的检测出，说明σ空间对于该图片比较合适。同时为了消除边缘响应（参考博客：https://blog.csdn.net/zddblog/article/details/7521424），一个定义不好的高斯差分算子的极值在横跨边缘的地方有较大的主曲率，而在垂直边缘的方向有较小的主曲率。DOG 算子会产生较强的边缘响应，需要剔除不稳定的边缘响应点。获取特征点处的 Hessian 矩阵，主曲率通过一个 2x2 的 Hessian 矩阵 H 求出：$\begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix}$, Tr(H) = $D_{xx}+D_{yy}$, Det(H) = $D_{xx}*D_{yy}-D_{xy}*D_{yx}$,为了剔除边缘响应点，需要让该比值小于一定的阈值，因此，为了检测主曲率是否在某域值 r 下，只需检测$\frac{Tr(H)\text{^}2}{Det(H)} < \frac{(r+1)\text{^}2}{r}$。代码中该阈值设置为 12.1。orientations 的计算则为高斯差分，将 math.atan2()的值域-pi 到 pi 分为 36 个段，将整个 blob 的所有像素均计算梯度，分配到之前的 36 个段，组成直方图。为了简化，将直方图中最大的 bin 对应的的弧度值作为该 blob 的主方向。

compute_descriptors 的计算方法为，将第一问的每个求出的 blob，在关键点尺度空间内 4*4 的窗口中计算的 8 个方向的梯度信息（即将梯度分成 8 个段的直方图并将其归一化后拼接起来），共 4*4*8=128 维向量表征。但是，为了保证旋转不变性，需要对于每个 blob 里的像素按照整个 blob 的主方向进行旋转（即将坐标轴进行统一）。



6.2 坐标轴旋转

图 2 坐标旋转示例

本次实验中所选取的半径（参考博客：https://blog.csdn.net/zddblog/article/details/7521424）为 2*$\sqrt{2} * \sqrt{3\sigma}$。

match_descriptors：输入为两张图片的 descriptors 信息，需要使用 l2-norm 来计算它们之间的相似度。同时对于某一个 descriptor 需要保证其与另外一张图片的 descriptors 中的最佳匹配与次佳匹配的 l2-norm 比值小于某一阈值，才能认为该匹配是成功的。根据老师的 slides 里的经验建议，选取的阈值为 0.7。具体实现上，构建了一个二维的矩阵（长宽分别是两张图片的 descriptor 的数量），用于保存其 l2-norm 的值，最后使用 np.argsort 函数进行排序，找到最佳匹配与次佳匹配的 l2-norm 比值。

draw_matches：如果输入是 outlier 则颜色变量设置为红色，否则设置为蓝色。如果两张图片的高度不同，则首先将高度较小的图片的高度 pad 到另一张图片的高度。接下来使用 np.concatenate 函数将两张图片拼接在一起。接下来遍历所有的匹配，用 cv2.line 以及 cv2.circle 函数画出匹配的点和线。

compute_affine_xform：使用 RANSAC 算法，即假设验证。由于我们需要拟合的是形如 $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ 的 affine transformation。其中共有 6 个未知数，一个匹配都能够得到 2 个方程，因此至少需要 3 个匹配来求解这个线性方程组。于是首先随机挑选 3 个匹配，利用 np.linalg.lstsq()函数求解线性方程组。重复上述实验 N 轮（这里设置为 1000 轮，为了避免小概率事件的发生），然后选择 inlier 数量（根据原坐标乘以变换矩阵后与新坐标之间的 l2-norm 小于阈值 13 来判断 inlier）最多的那组 affine transformation，作为最终的 transformation。

transformation matrices：

Bikes1-2：[[1.00940260e+00 -4.47746193e-03 3.58433650e-15]
 [5.33057583e-03 1.00891884e+00 -4.68158498e-15]
 [1.91580866e+01 -2.76157939e+01 1.00000000e+00]]

Bikes 1-3：[[1.00940260e+00 -4.47746193e-03 3.58433650e-15]
 [5.33057583e-03 1.00891884e+00 -4.68158498e-15]
 [1.91580866e+01 -2.76157939e+01 1.00000000e+00]]

leuven1-2：[[1.00088947e+00 4.86377517e-03 1.62474061e-18]
 [-7.60162549e-04 1.00018717e+00 -3.46944695e-18]
 [4.64834523e+00 -3.26184081e+00 1.00000000e+00]]

leuven1-3：[[1.00185718e+00 2.12070188e-04 9.24049572e-19]
 [3.24208735e-03 9.99658572e-01 -5.20417043e-18]
 [4.35192781e+00 -3.98250342e+00 1.00000000e+00]]

graf1-2:[[7.80063476e-01 -2.23279136e-01 1.66926787e-17]
 [2.57574731e-01 8.62113284e-01 -2.77555756e-17]
 [-8.28023300e+00 8.29493383e+01 1.00000000e+00]]

graf1-3:[[-7.22158279e-02 -8.22152721e-02 1.37230961e-17]
 [-1.62826230e-01 -5.09870760e-01 -1.04083409e-17]
 [3.20959584e+02 2.55125078e+02 1.00000000e+00]](效果不好)

Wall1-2:[[8.85166986e-01 2.76866280e-02 1.44931894e-15]

[3.08308074e-02 1.02154823e+00 -3.72705339e-15]
[-3.48912548e+01 1.44786872e+01 1.00000000e+00]]
Wall1-3：[[1.92199969e+00 1.26484537e+00 2.78893260e-03]
[-1.98077067e-02 1.90675648e-02 1.32258659e-03]
[5.33418101e-03 3.55282085e-03 9.52747112e-06]](效果不好)

stitch_images：使用 cv2.warpAffine 函数，输入 pad 之后的图像以及上一问得到的变换矩阵，可以直接得到拼接之后的图像，再对重合区域做一些处理就能得到最终的拼接图像。

以下是图像匹配以及拼接后的效果。

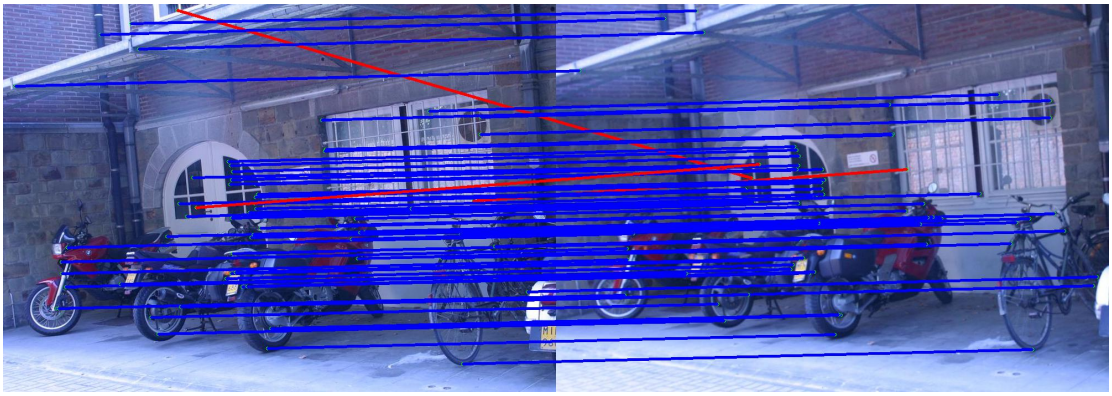


图 3bikes1-2 匹配结果

图 4bikes1-2 拼接结果

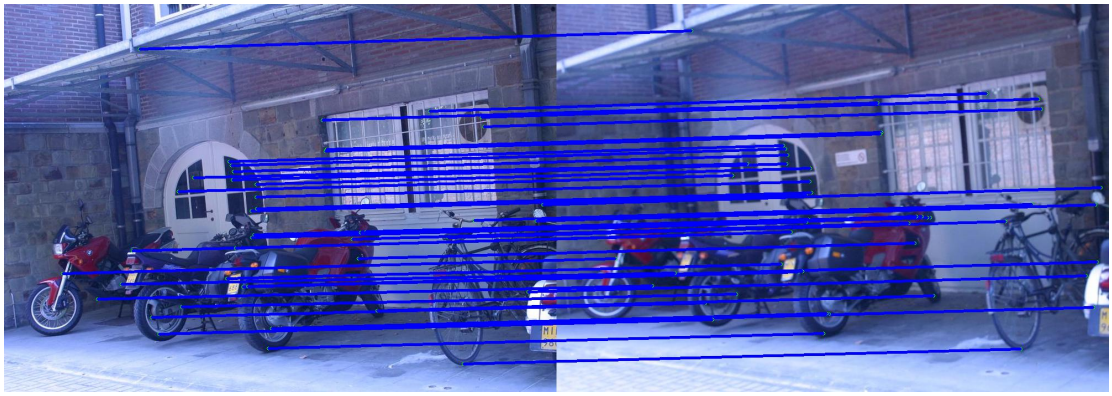


图 5bike1-3 匹配结果

图 6bike1-3 拼接结果

图 7graf1-2 匹配结果(参数：sig0 =0.2 s=1.2 len = 50)

图 8graf1-2 拼接结果

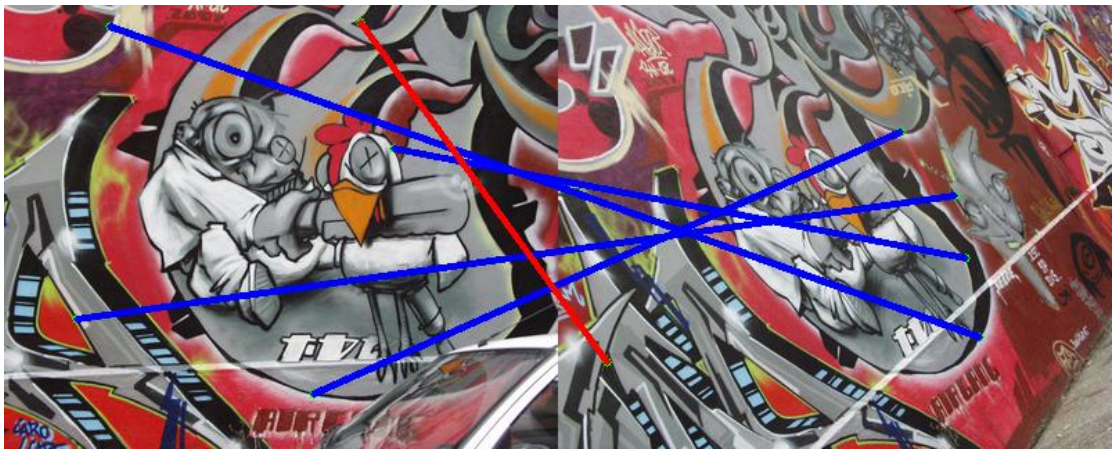


图 9graf1-3 拼接效果

图 10leuven1-2 匹配结果

图 11leuven1-2 拼接结果

图 12leuven1-3 匹配结果

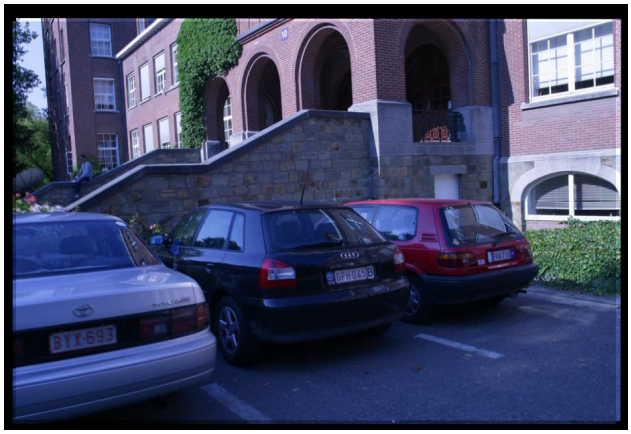


图 13leuven1-3 拼接结果

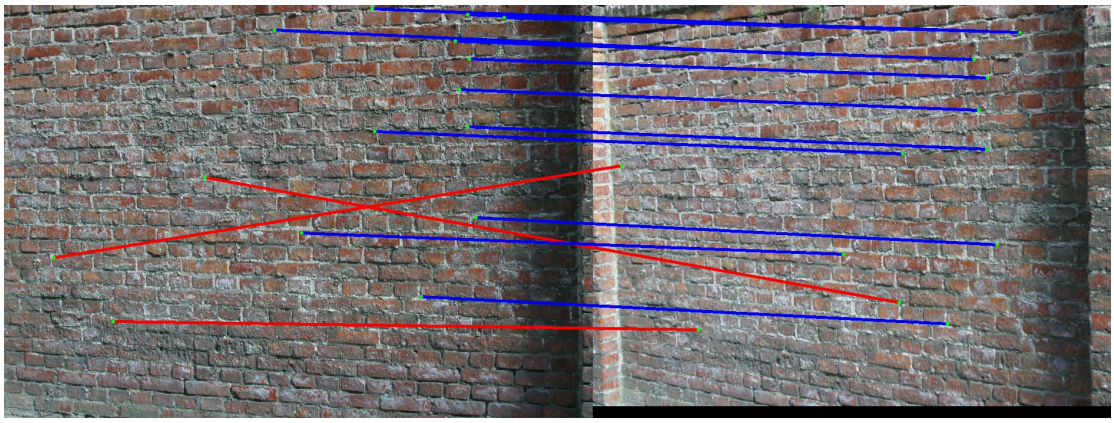


图 14wall1-2 匹配结果

图 15wall1-2 拼接结果

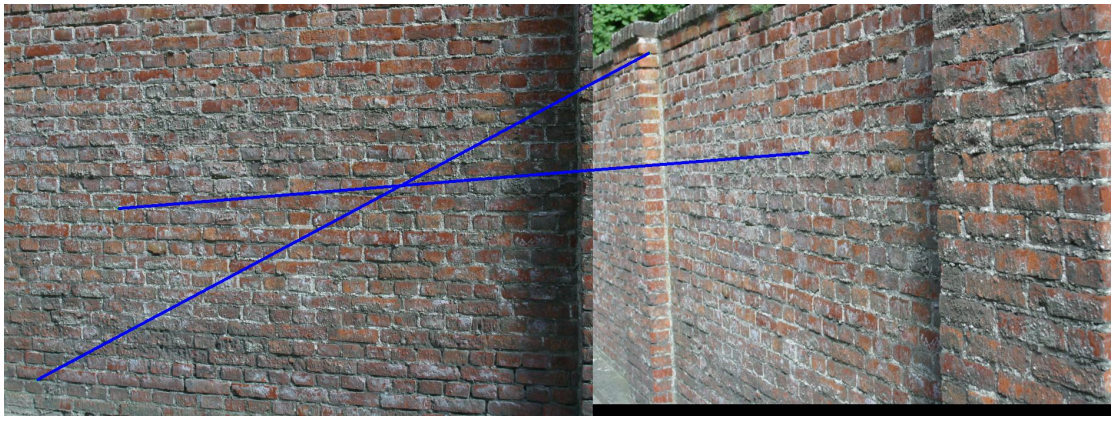


图 16wall1-3 匹配结果

graf1 和 graf3 的匹配数量较少导致无法拼接，个人认为原因在于 graf3 相对于 graf1 而言旋转角度太大，同时纹理非常复杂。旋转之后产生的近大远小也有尺度的变化。

wall1 和 wall3 的匹配数量也比较少，除了旋转角度过大之外，纹理过于相似导致在

match_descriptors 这一步里最佳匹配和次佳匹配的差距并不大，无法通过阈值筛选。