1. 程序设计
   1. 所编写的函数：
      1. SFD.describeFeatures(self, image, keypoints)
      2. MFD.describeFeatures(self, image, keypoints)
      3. SSD.matchFeatures(self, desc1, desc2)
      4. Ratio.matchFeatures(self, desc1, desc2)
   2. 详细算法描述
      1. SFD.describeFeatures函数

class SimpleFeatureDescriptor(FeatureDescriptor):

*# TODO: Implement parts of this function*

    def describeFeatures(*self*, *image*, *keypoints*):

        image = *image.astype(*np.float32*)*

        image /= 255.

        grayImage = *cv2.cvtColor(*image, cv2.COLOR\_BGR2GRAY*)*

        desc = *np.zeros(*(*len(*keypoints*)*, 5 \* 5)*)*

        height = image.shape[0]

        width = image.shape[1]

        newshape = (height + 4, width + 4)

        newimage = *np.zeros(*newshape, np.float32*)*

        newimage[2:-2, 2:-2] = grayImage

*for* i, f in *enumerate(*keypoints*)*:

            x, y = f.pt

            x, y = *int(*x*)*, *int(*y*)*

            nf = newimage[y:y+5, x:x+5]

            desc[i] = *nf.flatten()*

*return* desc

*# TODO 4: The simple descriptor is a 5x5 window of intensities*

*# sampled centered on the feature point. Store the descriptor*

*# as a row-major vector. Treat pixels outside the image as zero.*

*# TODO-BLOCK-BEGIN*

*raise* *Exception("TODO in features.py not implemented")*

*# TODO-BLOCK-END*

简单的特征描述算法，采用的是5x5的简单特征描述符，先将设置一个比原始图像扩大4个位置的零矩阵，将图像的灰度图填充其中，再以特征点为中心的5x5领域计算特征描述符，即该领域的灰度强度值，此简单描述符可以正常描述平移性质的图像。

* + 1. MFD.describeFeatures函数

class MOPSFeatureDescriptor(FeatureDescriptor):

*# TODO: Implement parts of this function*

    def describeFeatures(*self*, *image*, *keypoints*):

        image = *image.astype(*np.float32*)*

        image /= 255.

*# This image represents the window around the feature you need to*

*# compute to store as the feature descriptor (row-major)*

        windowSize = 8

        desc = *np.zeros(*(*len(*keypoints*)*, windowSize \* windowSize)*)*

        grayImage = *cv2.cvtColor(*image, cv2.COLOR\_BGR2GRAY*)*

        grayImage = *ndimage.gaussian\_filter(*grayImage, 0.5*)*

        zeroArray = *np.zeros(*(windowSize, windowSize), np.float32*)*

*for* i, f in *enumerate(*keypoints*)*:

*# TODO 5: Compute the transform as described by the feature*

*# location/orientation. You will need to compute the transform*

*# from each pixel in the 40x40 rotated window surrounding*

*# the feature to the appropriate pixels in the 8x8 feature*

*# descriptor image.*

            transMx = *np.zeros(*(2, 3)*)*

            x, y = f.pt

            x, y = *int(*x*)*, *int(*y*)*

*#newimage = grayImage[y-19:y+20, x-20:x+19]*

            angle = f.angle

            T1 = *trans.get\_trans\_mx\_2D(*-x, -y*)*

            R = *trans.get\_rot\_mx\_2D(*-angle \* np.pi / 180*)*

            S = *trans.get\_scale\_mx\_2D(*0.2, 0.2*)*

            T2 = *trans.get\_trans\_mx\_2D(*4, 4*)*

            T = *np.dot(*T2, *np.dot(*S, *np.dot(*R, T1*)))*

            transMx = T[:2, :]

*# TODO-BLOCK-BEGIN*

*#raise Exception("TODO in features.py not implemented")*

*# TODO-BLOCK-END*

*# Call the warp affine function to do the mapping*

*# It expects a 2x3 matrix*

            destImage = *cv2.warpAffine(*grayImage, transMx,(windowSize, windowSize), *flags*=cv2.INTER\_LINEAR*)*

            mean = *np.mean(*destImage*)*

            stdv = *np.std(*destImage*)*

*if*(stdv < 1e-5):

                desc[i] = *zeroArray.flatten()*

*else*:

                destImage = (destImage - mean) / stdv

                desc[i] = *destImage.flatten()*

*return* desc

*# TODO 6: Normalize the descriptor to have zero mean and unit*

*# variance. If the variance is zero then set the descriptor*

*# vector to zero. Lastly, write the vector to desc.*

*# TODO-BLOCK-BEGIN*

*raise* *Exception("TODO in features.py not implemented")*

*# TODO-BLOCK-END*

MOPS特征描述算法，采用了特征点周围子采样的8x8图像块，并采用cv2.warpAffine()函数进行特征块的转变，使其方向朝右。其中，变换矩阵采用平移、旋转、缩放、再平移的变换方法，形成2x3变换矩阵，为了使8x8图像块规范化，还对其进行了标准化的操作。

* + 1. SSD.matchFeatures函数

class SSDFeatureMatcher(FeatureMatcher):

    def matchFeatures(*self*, *desc1*, *desc2*):

        matches = []

*# feature count = n*

        assert desc1.ndim == 2

*# feature count = m*

        assert desc2.ndim == 2

*# the two features should have the type*

        assert desc1.shape[1] == desc2.shape[1]

*if* desc1.shape[0] == 0 or desc2.shape[0] == 0:

*return* []

        distances = *scipy.spatial.distance.cdist(*desc1, desc2, *metric* = *"euclidean")*

        mindist = *np.argmin(*distances, *axis* = 1*)*

*for* i, idx in *enumerate(*mindist*)*:

            match = *cv2.DMatch(*i, idx, distances[i, idx]*)*

*matches.append(*match*)*

*return* matches

*# TODO 7: Perform simple feature matching.  This uses the SSD*

*# distance between two feature vectors, and matches a feature in*

*# the first image with the closest feature in the second image.*

*# Note: multiple features from the first image may match the same*

*# feature in the second image.*

*# TODO-BLOCK-BEGIN*

*raise* *Exception("TODO in features.py not implemented")*

*# TODO-BLOCK-END*

SSD特征匹配算法，使计算两张比对图中对应特征点之间的欧氏距离，来进行特征匹配。

SSD计算式如下：

IMG_256

代码中，通过scipy.spatial.distance.cdist方法，快速实现原图与训练图特征点之间的距离计算，并以二维矩阵存储，通过np.argmin方法找出矩阵在行方向上的最小值，并将其角标存储在新矩阵中，至此，新矩阵mindist中，行代表原图特征点位置，而矩阵中的值则为训练图代表的位置。通过cv2.DMatch类存放原图特征点位置、训练图特征点位置、两点之间的欧氏距离，以match对象返回。

* + 1. Ratio.matchFeatures函数

class RatioFeatureMatcher(FeatureMatcher):

    def matchFeatures(*self*, *desc1*, *desc2*):

        matches = []

*# feature count = n*

        assert desc1.ndim == 2

*# feature count = m*

        assert desc2.ndim == 2

*# the two features should have the type*

        assert desc1.shape[1] == desc2.shape[1]

*if* desc1.shape[0] == 0 or desc2.shape[0] == 0:

*return* []

        dist = *scipy.spatial.distance.cdist(*desc1, desc2, *metric* = *"euclidean")*

        argdist = *np.argsort(*dist, *axis* = 1*)*

        length = dist.shape[0]

*for* i in *range(*length*)*:

            ratiodist = *float (*dist[i][argdist[i][0]] / dist[i][argdist[i][1]]*)*

            match = *cv2.DMatch(*i, argdist[i][0], ratiodist*)*

*matches.append(*match*)*

*return* matches

*# TODO 8: Perform ratio feature matching.*

*# This uses the ratio of the SSD distance of the two best matches*

*# and matches a feature in the first image with the closest feature in the*

*# second image.*

*# Note: multiple features from the first image may match the same*

*# feature in the second image.*

*# You don't need to threshold matches in this function*

*# TODO-BLOCK-BEGIN*

*raise* *Exception("TODO in features.py not implemented")*

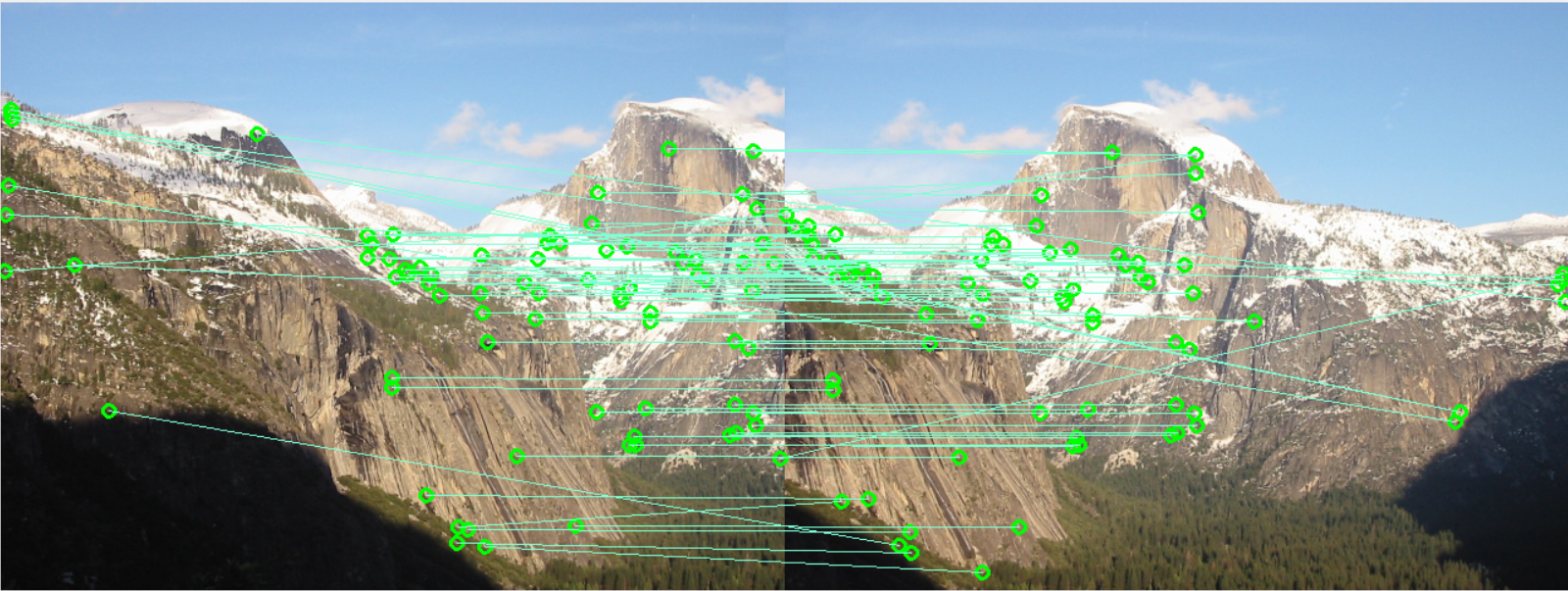
*# TODO-BLOCK-END*

Ratio比率距离匹配算法，与SSD算法实现类似，只是其中的距离计算不同，在次过程中，我依旧采用scipy.spatial.distance.cdist方法实现了欧氏距离的计算，但在之后，我使用np.argsort方法对矩阵进行了行方向上的从小到大的排序，找出最小距离与次小距离，两者相比得出比率距离，继续以match类对象返回。

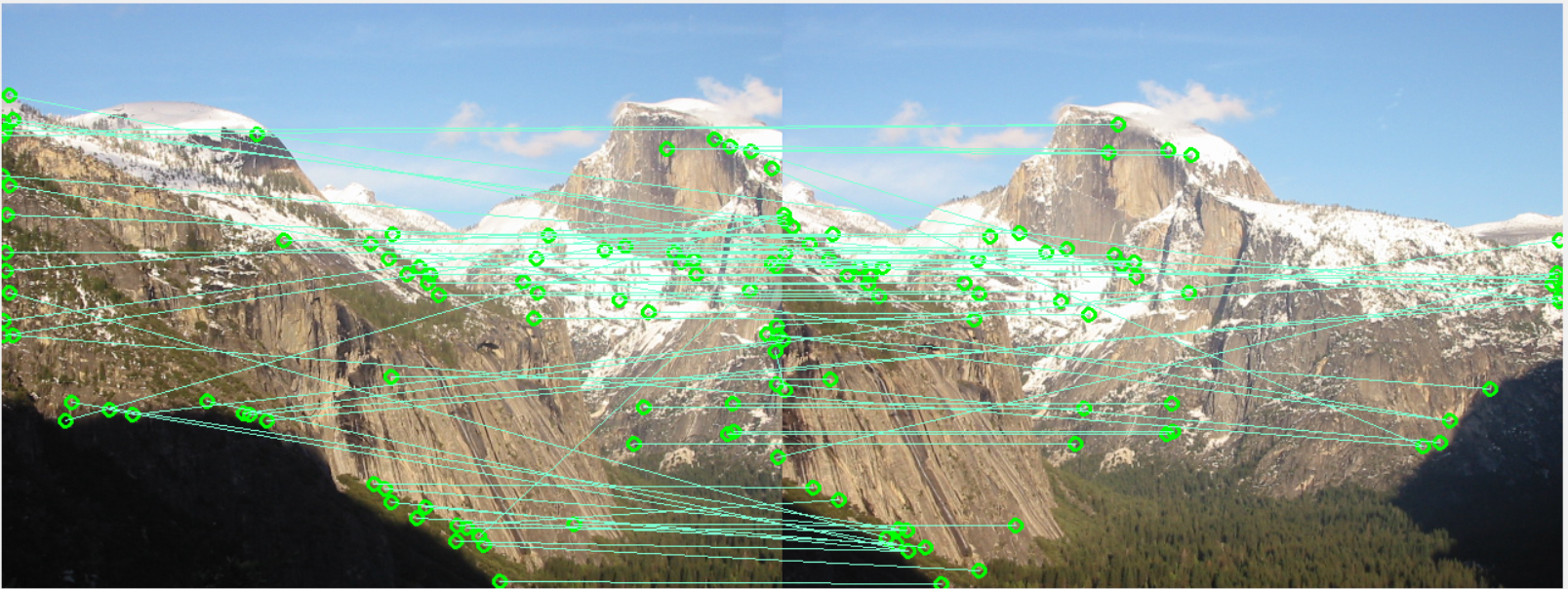
1. 实验结果



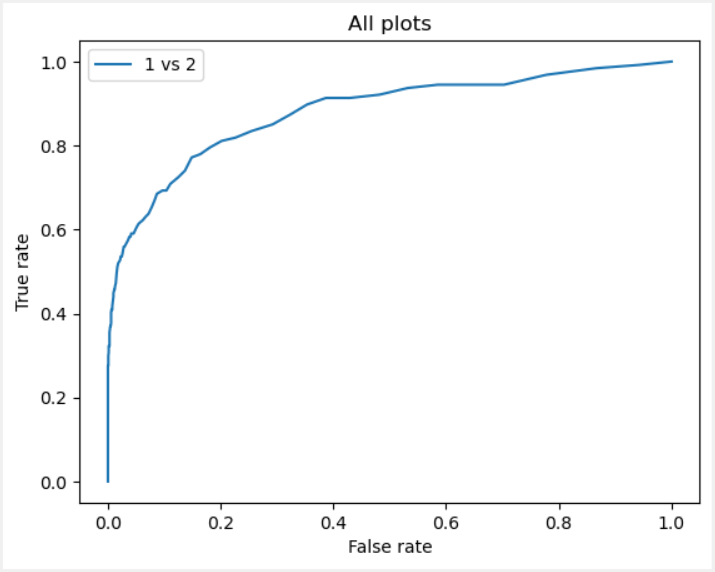
此图为Harris角点检测后保存的检测图，我们以resources\yosemite文件夹下的yosemite2.jpg为例，作出角点检测特征点图像。



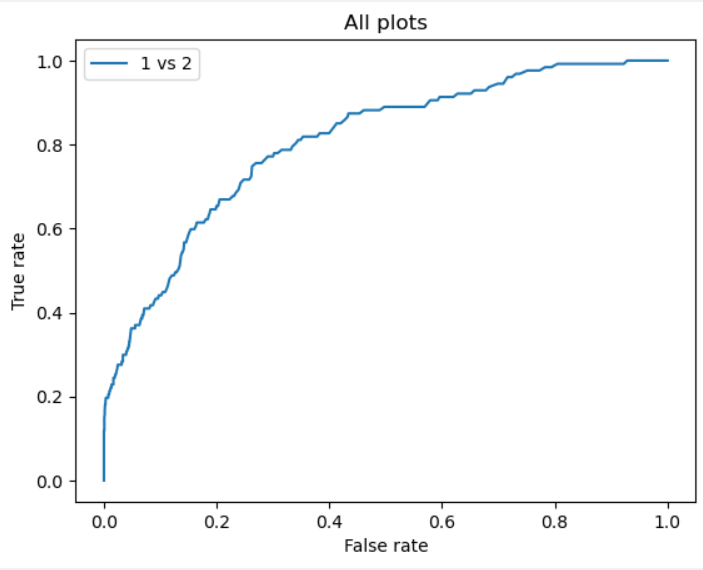
该图为通过Ratio比率测试测得的特征匹配图，共匹配到1554个特征点。



该图为通过SSD测试得到的特征匹配图。



该图为Ratio测试得到的ROC曲线，AUC值近似为0.880，Threshold值为10^(-2)。



该图为SSD测试得到的ROC曲线，AUC值近似为0.806，Threshold值为10^(-2)。

通过上述两种匹配方法的测试，可以得出，Ratio比率距离测试要优于SSD距离测试，

Ratio算法的AUC更大，效果更好。