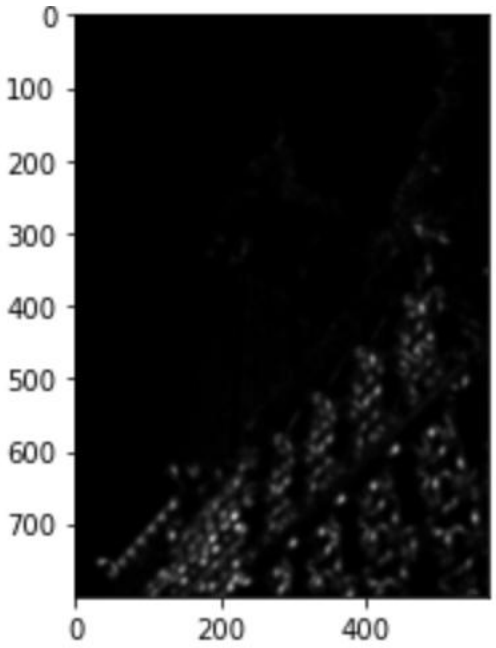
实验 2 局部图像描述子实验

编写代码完成以下实验内容，请将代码和运行效果截图插入到问题下方。

第一部分 Harris 角点检测器

1、 Harris 角点检测算法是通过检测像素周围是否存在两个或以上的不同方向的边。如果存在，则该点为角点。根据 Harris 角点检测器相关公式，编写代码实现 Harris 角点响应函数 compute_harris_response(im,sigma)。函数中可以使用 scipy.ndimage.filters 模块中的高斯导数滤波器来计算导数。参数 sigma 用来定义高斯滤波器的尺度大小，尝试不同的 sigma 值来取得最佳效果。该函数的返回值为 Harris 响应函数值的一幅图像。将图像“empire.jpg”作为参数，并设置适当的尺度参数，调用函数 compute_harris_response，并将返回的响应函数图像输出。

```
from scipy.ndimage import filters
from PIL import Image
from pylab import *
from numpy import *
import os

def compute_harris_response(im, sigma=3):
    """ 在一幅灰度图像中，对每个像素计算 Harris 角点检测器响应函数 """
    # 计算导数
    imx = zeros(im.shape)
    filters.gaussian_filter(im, (sigma, sigma), (0, 1), imx)
    imy = zeros(im.shape)
    filters.gaussian_filter(im, (sigma, sigma), (1, 0), imy)

    # 计算 Harris 矩阵的分量
    Wxx = filters.gaussian_filter(imx * imx, sigma)
    Wxy = filters.gaussian_filter(imx * imy, sigma)
    Wyy = filters.gaussian_filter(imy * imy, sigma)

    # 计算特征值和迹
    Wdet = Wxx * Wyy - Wxy ** 2
    Wtr = Wxx + Wyy

    return Wdet / Wtr

im = array(Image.open('../pics2/empire.jpg').convert('L'))
harrisim = compute_harris_response(im)
imshow(harrisim)
show()
```



2、 Harris 角点检测器仅仅能够检测出图像中的兴趣点，但无法通过比较图像间的兴趣点来寻找匹配角点的方法，需要在每个点上加入描述子信息，并给出一个比较描述子的方法。

兴趣点描述子是分配给兴趣点的一个向量，描述该点附近的图像的表观信息。描述子越好，寻找到的对应点就越好。我们用对应点来描述相同物体和场景点在不同图像上形成的像素点。

Harris 角点的描述子通常是由周围图像像素块的灰度值，以及用于比较的归一化互相关矩阵构成的。图像的像素块由以该像素点为中心的周围矩形部分图像构成。

为获取图像像素块，并使用归一化的互相关矩阵来比较它们，请编写如下两个函数：

（1） 函数 get_descriptors(image,filtered_coords,wid=5)：输入一副图像，对于每个返回的点，返回点周围 2*wid+1 个像素。

（2） 函数 match(desc1,desc2,threshold=0.5)：对于第一幅图像中的每个角点描述子，使用归一化互相关，选取它在第二幅图像中的匹配角点。

编写函数 match_twosided(desc1,desc2,threshold=0.5)实现两幅图像的匹配：从第二幅图像向第一幅图像匹配，然后过滤掉在两种方法中不是最好的匹配。

编写函数实现将匹配结果显示出来，可以通过在两边分别绘制出图像，然后使用线

段连接匹配的像素点来直观地可视化。可以将该任务分解为两个函数来实现：

（1） 函数 appendimages(im1,im2)：将两幅图像并排拼接成一幅新图像

（2） 函数 plot_matches(im1,im2,locs1,locs2,matchscores,show_below=True)：显示一幅带有连接匹配点连线的图片。其中，im1,im2 为数组图像，locs1,locs2 为特征位置。

通过编写代码和调用上述函数，实现将两幅不同尺度和角度的图像“crans_1_small.jpg”、“crans_2_small.jpg”实现 Harris 角点匹配，并将两幅图像中对应的匹配点用直线连接标示出来。将以上结果显示出来。

提示：由于计算资源的限制，如果图像比较大，可以通过调整图像尺寸的方式来降低像素数。

In [3]:
```
def get_harris_points(harrisim, min_dist=10, threshold=0.1):  # threshold阈值
    """ 从一幅 Harris 响应图像中返回角点。min_dist 为分割角点和图像边界的最少像素数目 """
    # 寻找高于阈值的候选角点
    corner_threshold = harrisim.max() * threshold
    harrisim_t = (harrisim > corner_threshold) * 1
    # 得到候选点的坐标
    coords = array(harrisim_t.nonzero()).T
    # 以及它们的 Harris 响应值
    candidate_values = [harrisim[c[0], c[1]] for c in coords]
    # 对候选点按照 Harris 响应值进行排序
    index = argsort(candidate_values)
    # 将可行点的位置保存到数组中
    allowed_locations = zeros(harrisim.shape)
    allowed_locations[min_dist:-min_dist, min_dist:-min_dist] = 1
    # 按照 min_distance 原则，选择最佳 Harris 点
    filtered_coords = []
    for i in index:
        if allowed_locations[coords[i, 0], coords[i, 1]] == 1:
            filtered_coords.append(coords[i])
            allowed_locations[(coords[i, 0] - min_dist):(coords[i, 0] + min_dist), \
            (coords[i, 1] - min_dist):(coords[i, 1] + min_dist)] = 0
    return filtered_coords


def plot_harris_points(image, filtered_coords):
    """ 绘制图像中检测到的角点 """
    figure()
    gray()
    imshow(image)
    plot([p[1] for p in filtered_coords], [p[0] for p in filtered_coords], '*')
    axis('off')
    show()
```

```
def get_descriptors(image, filtered_coords, wid=5):
    """ 对于每个返回的点，返回点周围 2*wid+1 个像素的值（假设选取点的 min_distance > wid）"""
    desc = []
    for coords in filtered_coords:
        patch = image[coords[0] - wid:coords[0] + wid + 1, coords[1] - wid:coords[1] + wid + 1].flatten()
        desc.append(patch)
    return desc


def match(desc1, desc2, threshold=0.5):
    """ 对于第一幅图像中的每个角点描述子，使用归一化互相关，选取它在第二幅图像中的匹配角点 """
    n = len(desc1[0])
    # 点对的距离
    d = -ones((len(desc1), len(desc2)))
    for i in range(len(desc1)):
        for j in range(len(desc2)):
            d1 = (desc1[i] - mean(desc1[i])) / std(desc1[i])
            d2 = (desc2[j] - mean(desc2[j])) / std(desc2[j])
            ncc_value = sum(d1 * d2) / (n - 1)
            if ncc_value > threshold:
                d[i, j] = ncc_value
    ndx = argsort(-d)
    matchscores = ndx[:, 0]
    return matchscores


def match_twosided(desc1, desc2, threshold=0.5):
    """ 两边对称版本的 match()"""
    matches_12 = match(desc1, desc2, threshold)
    matches_21 = match(desc2, desc1, threshold)
    ndx_12 = where(matches_12 >= 0)[0]
    # 去除非对称的匹配
    for n in ndx_12:
```

```
def match_twosided(desc1, desc2, threshold=0.5):
    """ 两边对称版本的 match()"""
    matches_12 = match(desc1, desc2, threshold)
    matches_21 = match(desc2, desc1, threshold)
    ndx_12 = where(matches_12 >= 0)[0]
    # 去除非对称的匹配
    for n in ndx_12:
        if matches_21[matches_12[n]] != n:
            matches_12[n] = -1
    return matches_12


def appendimages(im1, im2):
    """ 返回将两幅图像并排拼接成的一幅新图像 """
    # 选取具有最少行数的图像，然后填充足够的空行
    rows1 = im1.shape[0]
    rows2 = im2.shape[0]
    if rows1 < rows2:
        im1 = concatenate((im1, zeros((rows2 - rows1, im1.shape[1]))), axis=0)
    elif rows1 > rows2:
        im2 = concatenate((im2, zeros((rows1 - rows2, im2.shape[1]))), axis=0)
    # 如果这些情况都没有，那么它们的行数相同，不需要进行填充
    return concatenate((im1, im2), axis=1)


def plot_matches(im1, im2, locs1, locs2, matchscores, show_below=True):
    """ 显示一幅带有连接匹配之间连线的图片
   输入：im1，im2（数组图像），locs1，locs2（特征位置），matchscores（match() 的输出），
    show_below（如果图像应该显示在匹配的下方）"""
    im3 = appendimages(im1, im2)
    if show_below:
        im3 = vstack((im3, im3))
    imshow(im3)
```

```
def plot_matches(im1, im2, locs1, locs2, matchscores, show_below=True):
    """ 显示一幅带有连接匹配之间连线的图片
   输入：im1，im2（数组图像），locs1，locs2（特征位置），matchscores（match() 的输出），
    show_below（如果图像应该显示在匹配的下方）"""
    im3 = appendimages(im1, im2)
    if show_below:
        im3 = vstack((im3, im3))
    imshow(im3)
    cols1 = im1.shape[1]
    for i, m in enumerate(matchscores):
        if m > 0:
            plot([locs1[i][1], locs2[m][1] + cols1], [locs1[i][0], locs2[m][0]], 'c')
    axis('off')


wid = 5
im1 = array(Image.open('../pics2/crans_1_small.jpg').convert('L'))
im2 = array(Image.open('../pics2/crans_2_small.jpg').convert('L'))
harrisim = compute_harris_response(im1,5)
filtered_coords1 = get_harris_points(harrisim,wid+1)
d1 = get_descriptors(im1,filtered_coords1,wid)
harrisim = compute_harris_response(im2,5)
filtered_coords2 = get_harris_points(harrisim,wid+1)
d2 = get_descriptors(im2,filtered_coords2,wid)
print('starting matching')
matches = match_twosided(d1,d2)
figure()
gray()
plot_matches(im1,im2,filtered_coords1,filtered_coords2,matches)
show()
```

第二部分 SIFT 尺度不变特征变换

SIFT（尺度不变特征变换）是非常经典的图像局部描述子。SIFT 特征对于尺度、旋转和亮度都具有不变性。相关内容请参考教材和文献。

检测兴趣点

1、 这里可以使用开源工具包 VLFeat 提供的二进制文件来计算图像的 SIFT 特征。在 http://www.vlfeat.org/下载 VLFeat 工具包，解压后，只需根据操作系统选择相应的二进制文件夹将 sift.exe、vl.dll、vl.lib 拷贝到项目文件夹下。注意 VLFeat 版本要选择 0.9.20，最新版本不稳定。

2、 编写函数 process_image 用来调用 VLFeat 中的 sift.exe, 生成图像的 SIFT 特征文件。参考代码如下。

```
def process_image(imagename,resultname,params="--edge-thresh 10 --peak-thresh 5"):
    """ Process an image and save the results in a file. """

    if imagename[-3:] != 'pgm':
        # create a pgm file
        im = Image.open(imagename).convert('L')
        im.save('tmp.pgm')
        imagename = 'tmp.pgm'

    cmmd = str("sift "+imagename+" --output="+resultname+
                " "+params)
    os.system(cmmd)
    print('processed', imagename, 'to', resultname)
```

3、 编写函数 read_features_from_file，实现从函数 process_image 的输出文件中读取特征到 NumPy 数组中。这里是通过 NumPy 库中的 loadtxt()函数来完成。

```
def read_features_from_file(filename):
    """ Read feature properties and return in matrix form. """

    f = loadtxt(filename)
    return f[:,:4],f[:,4:] # feature locations, descriptors
```

4、 下一步需要将输出结果保存到特征文件中, 可以调用 NumPy 库中的 savetxt()函数来实现。编写函数 write_features_to_file 实现该功能。参考代码如下, 这里使用了 hstack()函数，该函数通过拼接不同的行向量来实现水平堆叠两个向量的功能。

```
def write_features_to_file(filename,locs,desc):
    """ Save feature location and descriptor to file. """
    savetxt(filename,hstack((locs,desc)))
```

5、 读取特征后，通过在图像上绘制出它们的位置，并将其可视化。编写函数实现在原始图像上使用蓝色的圆圈绘制出 SIFT 特征点的位置。

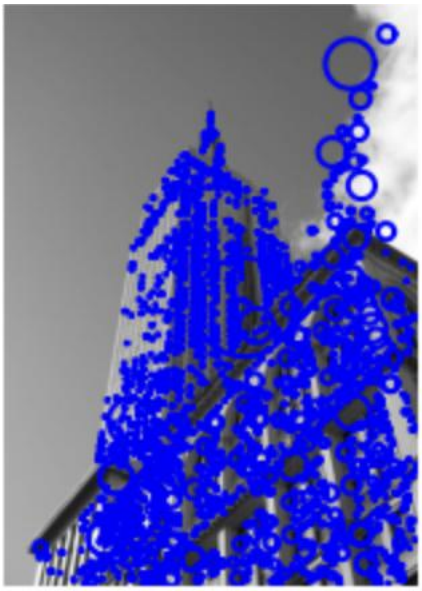
```
def plot_features(im,locs,circle=False):
    """ Show image with features. input: im (image as array),
        locs (row, col, scale, orientation of each feature). """

    def draw_circle(c,r):
        t = arange(0,1.01,.01)*2*pi
        x = r*cos(t) + c[0]
        y = r*sin(t) + c[1]
        plot(x,y,'b',linewidth=2)

    imshow(im)
    if circle:
        for p in locs:
            draw_circle(p[:2],p[2])
    else:
        plot(locs[:,0],locs[:,1],'ob')
    axis('off')
```

6、 调用已经编写的 SIFT 相关函数，对于图像“empire.jpg”进行 SIFT 特征分析，并将结果绘制出来。

In [8]:

```
def process_image(imagename, resultname, params="--edge-thresh 10 --peak-thresh 5"):
    """ 处理一幅图像，然后将结果保存在文件中 """
    if imagename[-3:] != 'pgm':
        # 创建一个 pgm 文件
        im = Image.open(imagename).convert('L')
        im.save('tmp.pgm')
        imagename = 'tmp.pgm'
    cmmd = str("sift " + imagename + " --output=" + resultname + " " + params)
    os.system(cmmd)
    print('processed', imagename, 'to', resultname)


def read_features_from_file(filename):
    """ 读取特征属性值，然后将其以矩阵的形式返回 """
    f = loadtxt(filename)
    return f[:, :4], f[:, 4:]  # 特征位置，描述子


def write_features_to_file(filename, locs, desc):
    """ 将特征位置和描述子保存到文件中 """
    savetxt(filename, hstack((locs, desc)))


def plot_features(im, locs, circle=False):
    """ 显示带有特征的图像
   输入：im（数组图像），locs（每个特征的行、列、尺度和朝向）"""

    def draw_circle(c, r):
        t = arange(0, 1.01, .01) * 2 * pi
        x = r * cos(t) + c[0]
        y = r * sin(t) + c[1]
        plot(x, y, 'b', linewidth=2)
```

```
def plot_features(im, locs, circle=False):
    """ 显示带有特征的图像
   输入：im（数组图像），locs（每个特征的行、列、尺度和朝向）"""

    def draw_circle(c, r):
        t = arange(0, 1.01, .01) * 2 * pi
        x = r * cos(t) + c[0]
        y = r * sin(t) + c[1]
        plot(x, y, 'b', linewidth=2)
        imshow(im)

    if circle:
        for p in locs:
            draw_circle(p[:2], p[2])
    else:
        plot(locs[:, 0], locs[:, 1], 'ob')
    axis('off')


imname = 'empire.jpg'
im1 = array(Image.open('../pics2/empire.jpg').convert('L'))
process_image('../pics2/empire.jpg','../pics2/empire.sift')
l1,d1 = read_features_from_file('../pics2/empire.sift')
figure()
gray()
plot_features(im1,l1,circle=True)
show()
```



匹配描述子

7、 对于将一幅图像中的特征匹配到另一幅图像的特征，一种稳健的准则是使用这两个特征距离和两个最匹配特征距离的比率。使用该方法可以使错误的匹配数降低。实现匹配函数 match，对于第一幅图像中的每个描述子，选取其在第二幅图像中的匹配。

```
def match(desc1,desc2):
    """ For each descriptor in the first image,
        select its match in the second image.
        input: desc1 (descriptors for the first image),
        desc2 (same for second image). """

    desc1 = array([d/linalg.norm(d) for d in desc1])
    desc2 = array([d/linalg.norm(d) for d in desc2])

    dist_ratio = 0.6
    desc1_size = desc1.shape

    matchscores = zeros((desc1_size[0]),'int')
    desc2t = desc2.T # precompute matrix transpose
    for i in range(desc1_size[0]):
        dotprods = dot(desc1[i,:],desc2t) # vector of dot products
        dotprods = 0.9999*dotprods
        # inverse cosine and sort, return index for features in second image
        indx = argsort(arccos(dotprods))

        # check if nearest neighbor has angle less than dist_ratio times 2nd
        if arccos(dotprods)[indx[0]] < dist_ratio * arccos(dotprods)[indx[1]]:
            matchscores[i] = int(indx[0])

    return matchscores
```

8、 为了增加匹配的稳健性，再反过来执行一次匹配步骤，即从第二幅图像中的特征向第一幅图像中的特征匹配。最后，仅仅保留同时满足两种匹配准则的对应点。编写代码实现该操作。

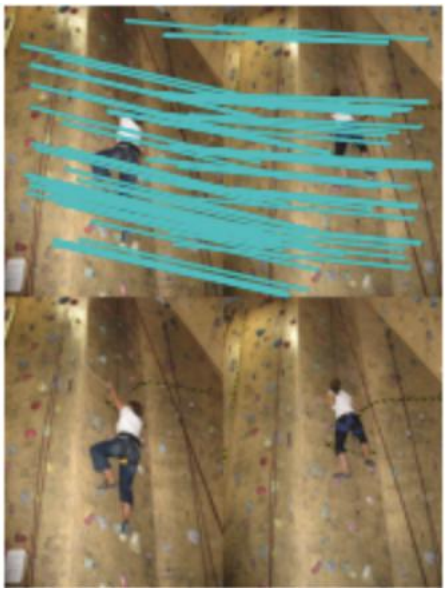
```
def match_twosided(desc1,desc2):
    """ Two-sided symmetric version of match(). """

    matches_12 = match(desc1,desc2)
    matches_21 = match(desc2,desc1)

    ndx_12 = matches_12.nonzero()[0]

    # remove matches that are not symmetric
    for n in ndx_12:
        if matches_21[int(matches_12[n])] != n:
            matches_12[n] = 0

    return matches_12
```

9、 用 SIFT 方法对图像“climbing_1_small.jpg”、“climbing_2_small.jpg”进行特征点检测，并进行两幅图像的特征点匹配。然后，在两幅图像上，将对应的特征点用直线连接起来。编码调用已编写的函数实现上述功能。注意，在绘制图像时，可以直接使用前面已经编写好的 appendimages()和 plot_matches()函数。

In [18]:
```
def match2(desc1, desc2):
    """ 对于第一幅图像中的每个描述子，选取其在第二幅图像中的匹配
    输入：desc1（第一幅图像中的描述子），desc2（第二幅图像中的描述子）"""
    desc1 = array([d / linalg.norm(d) for d in desc1])
    desc2 = array([d / linalg.norm(d) for d in desc2])

    dist_ratio = 0.6
    desc1_size = desc1.shape

    matchscores = zeros((desc1_size[0], 1), 'int')
    desc2t = desc2.T  # 预先计算矩阵转置
    for i in range(desc1_size[0]):
        dotprods = dot(desc1[i, :], desc2t)  # 向量点乘
        dotprods = 0.9999 * dotprods
        # 反余弦和反排序，返回第二幅图像中特征的索引
        indx = argsort(arccos(dotprods))
        # 检查最近邻的角度是否小于 dist_ratio 乘以第二近邻的角度
        if arccos(dotprods)[indx[0]] < dist_ratio * arccos(dotprods)[indx[1]]:
            matchscores[i] = int(indx[0])
    return matchscores


def match_twosided2(desc1, desc2):
    """ 双向对称版本的 match()"""
    matches_12 = match2(desc1, desc2)
    matches_21 = match2(desc2, desc1)
    ndx_12 = matches_12.nonzero()[0]
    # 去除不对称的匹配
    for n in ndx_12:
        if matches_21[int(matches_12[n])] != n:
            matches_12[n] = 0
    return matches_12
```

```
            matches_12[n] = 0
    return matches_12


def appendimages(im1, im2):
    """ 返回将两幅图像并排拼接成的一幅新图像 """
    # 选取具有最少行数的图像，然后填充足够的空行
    rows1 = im1.shape[0]
    rows2 = im2.shape[0]
    if rows1 < rows2:
        im1 = concatenate((im1, zeros((rows2 - rows1, im1.shape[1]))), axis=0)
    elif rows1 > rows2:
        im2 = concatenate((im2, zeros((rows1 - rows2, im2.shape[1]))), axis=0)
    # 如果这些情况都没有，那么它们的行数相同，不需要进行填充
    return concatenate((im1, im2), axis=1)


def plot_matches(im1, im2, locs1, locs2, matchscores, show_below=True):
    """ 显示一幅带有连接匹配之间连线的图片
   输入：im1，im2（数组图像），locs1，locs2（特征位置），matchscores（match() 的输出），
    show_below（如果图像应该显示在匹配的下方）"""
    im3 = appendimages(im1, im2)
    if show_below:
        im3 = vstack((im3, im3))
    imshow(im3)
    cols1 = im1.shape[1]
    for i, m in enumerate(matchscores):
        if m > 0:
            plot([locs1[i][0], locs2[int(m)][0] + cols1], [locs1[i][1], locs2[int(m)][1]], 'c')
    axis('off')
```

```
im1 = array(Image.open('../pics2/climbing_1_small.jpg'))
im2 = array(Image.open('../pics2/climbing_2_small.jpg'))
process_image('../pics2/climbing_1_small.jpg','../pics2/climbing_1_small.jpg.sift')
process_image('../pics2/climbing_2_small.jpg','../pics2/climbing_2_small.jpg.sift')
l1,d1 = read_features_from_file('../pics2/climbing_1_small.jpg.sift')
l2,d2 = read_features_from_file('../pics2/climbing_2_small.jpg.sift')
print('starting matching')
matches = match_twosided2(d1,d2)
figure()
plot_matches(im1,im2,l1,l2,matches)
show()
```



提高层次

1、为了让匹配具有更强的稳健性，修改用于匹配 Harris 角点的函数，使其输入参数中包含认为两点存在对应关系允许的最大像素距离。

In [21]:

```
def my_match(desc1, desc2, threshold=0.5, max=None):
    n = len(desc1[0])
    # 点对的距离
    d = -ones((len(desc1), len(desc2)))
    for i in range(len(desc1)):
        for j in range(len(desc2)):
            d1 = (desc1[i] - mean(desc1[i])) / std(desc1[i])
            d2 = (desc2[j] - mean(desc2[j])) / std(desc2[j])
            ncc_value = sum(d1 * d2) / (n - 1)
            if ncc_value > threshold and ncc_value <= max :
                d[i, j] = ncc_value
    ndx = argsort(-d)
    matchscores = ndx[:, 0]
    return matchscores
```

```
def my_match_twosided(desc1, desc2, threshold=0.5):
    """ 两边对称版本的 match()"""
    # max = 0.7
    matches_12 = my_match(desc1, desc2, threshold, 0.7)
    matches_21 = my_match(desc2, desc1, threshold, 0.7)
    ndx_12 = where(matches_12 >= 0)[0]
    # 去除非对称的匹配
    for n in ndx_12:
        if matches_21[matches_12[n]] != n:
            matches_12[n] = -1
    return matches_12

wid = 5
im1 = array(Image.open('../pics2/crans_1_small.jpg').convert('L'))
im2 = array(Image.open('../pics2/crans_2_small.jpg').convert('L'))
harrisim = compute_harris_response(im1,5)
filtered_coords1 = get_harris_points(harrisim,wid+1)
d1 = get_descriptors(im1,filtered_coords1,wid)
harrisim = compute_harris_response(im2,5)
filtered_coords2 = get_harris_points(harrisim,wid+1)
d2 = get_descriptors(im2,filtered_coords2,wid)
print('starting matching')
matches = my_match_twosided(d1,d2)
figure()
gray()
plot_matches(im1,im2,filtered_coords1,filtered_coords2,matches)
show()
```

```
starting matching
```

2、对一幅图像不断地应用模糊操作，使得模糊效果越来越强，然后提取 Harris 角点，会出现什么问题?

```
65      def My_harris(img):
66          gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
67          dst = cv2.cornerHarris(gray, 3, 23, 0.04)
68          img[dst > 0.01 * dst.max()] = [0, 0, 255]
69          cv2.imshow(' ', img)
70          cv2.waitKey(0)
71          cv2.destroyAllWindows()
72
73
74      if __name__ == '__main__':
75          img2 = cv2.GaussianBlur(img1, (11, 11), 0)
76          img3 = cv2.GaussianBlur(img1, (31, 31), 0)
77          img4 = cv2.GaussianBlur(img1, (51, 51), 0)
78          img5 = cv2.GaussianBlur(img1, (71, 71), 0)
79          img6 = cv2.GaussianBlur(img1, (91, 91), 0)
80          img7 = cv2.GaussianBlur(img1, (111, 111), 0)
81          img8 = cv2.GaussianBlur(img1, (131, 131), 0)
82          img9 = cv2.GaussianBlur(img1, (151, 151), 0)
```

由上述图片可见，图像越模糊，提取 harris 角点时越容易发生误判，即提取角点数目增多。

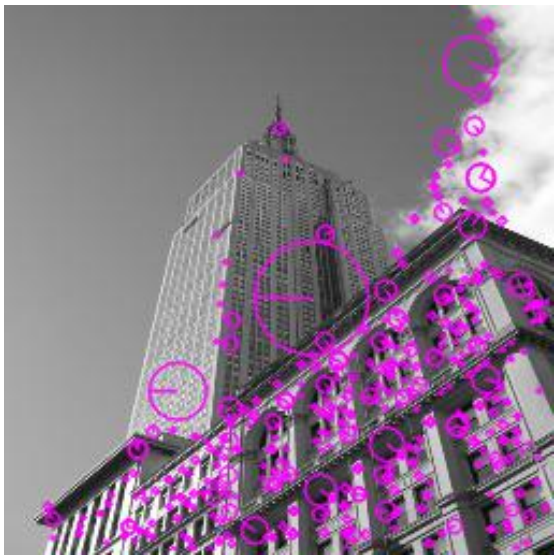
3、尝试使用另一种 Harris 角点检测器，即：快速角点检测器，使用敏感性的阈值，将检测结果与 Harris 角点检测结果进行比较。（快速角点检测器的可以在 http://www.edwardrosten.com/work/fast.html 下载）

```python
import cv2

image1 = cv2.imread('pics2/empire.jpg')



# img1 = cv2.resize(img1,dsize=(600,400))
# image1 = img1.copy()
def My_fast(image1):
    fast = cv2.FastFeatureDetector_create(threshold=50)
    keypoints1 = fast.detect(image1, None)
    # 在图像上绘制关键点
    image1 = cv2.drawKeypoints(image=image1, keypoints=keypoints1, outImage=image1, color=(255, 0, 255),\
                               flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    print('show')
    cv2.imshow('fast_keypoints1', image1)
    cv2.waitKey(0)

def My_orb(image):
    gray1 = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image1 = gray1.copy()
    orb = cv2.ORB_create(128)
    keypoints1, descriptors1 = orb.detectAndCompute(image1, None)
    image1 = cv2.drawKeypoints(image=image1, keypoints=keypoints1, outImage=image1, color=(255, 0, 255),\
                               flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    cv2.imshow('orb_keypoints1', image1)
    cv2.waitKey(0)

```

左图为 fast 角点检测，右图为 orb 角点检测,harris 角点检测见上题。

4、 以不同分辨率创建一幅图像的副本，例如：可以尝试多次将图像的尺寸减半。对每幅图像提取 SIFT 特征。绘制并匹配特征，说明尺度的独立性是如何失效的。

```
def My_sift(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    img = gray.copy()
    sift = cv2.xfeatures2d.SIFT_create(400)
    keypoints1, descriptor1 = sift.detectAndCompute(img, None)
    img = cv2.drawKeypoints(image=img, keypoints=keypoints1, outImage=img, color=(255, 0, 255),\
                            flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    cv2.imshow('sift_keypoints1', img)
    cv2.waitKey(0)

if __name__ == '__main__':
    img2 = cv2.resize(img1, dsize=(512, 512))
    img3 = cv2.resize(img1, dsize=(256, 256))
    img4 = cv2.resize(img1, dsize=(128, 128))
    img5 = cv2.resize(img1, dsize=(64, 64))
    img6 = cv2.resize(img1, dsize=(32, 32))
    My_sift(img6)
```



上图从左至右依次为正常图像,512*512,256*256,128*128,64*64,32*32
下图为匹配代码以及原图和各个压缩后的图像的匹配结果

```python
def My_sift(image1,image2):
    gray = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    img1 = gray.copy()
    gray = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
    img2 = gray.copy()
    sift = cv2.xfeatures2d.SIFT_create(400)
    keypoints1, descriptor1 = sift.detectAndCompute(img1, None)
    keypoints2, descriptor2 = sift.detectAndCompute(img2, None)
    img1 = cv2.drawKeypoints(image=img1, keypoints=keypoints1, outImage=img1, color=(255, 0, 255),\
                             flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    img2 = cv2.drawKeypoints(image=img2, keypoints=keypoints1, outImage=img2, color=(255, 0, 255),\
                             flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    matcher = cv2.FlannBasedMatcher()
    matchePoints = matcher.match(descriptor1, descriptor2)
    minMatch = 1
    maxMatch = 0
    for i in range(len(matchePoints)):
        if minMatch > matchePoints[i].distance:
            minMatch = matchePoints[i].distance
        if maxMatch < matchePoints[i].distance:
            maxMatch = matchePoints[i].distance
    goodMatchePoints = []
    for i in range(len(matchePoints)):
        if matchePoints[i].distance < minMatch + (maxMatch - minMatch) / 3:
            goodMatchePoints.append(matchePoints[i])
    outImg = None
    outImg = cv2.drawMatches(img1, keypoints1, img2, keypoints2, goodMatchePoints, outImg, matchColor=(0, 255, 0),
                             flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT)
    cv2.imshow('matche', outImg)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

在不断对图像进行折半压缩的过程中，角点的检验的独立性随着图像的模糊程度增高二缺失，知道图像压缩为 32*32 后，独立性完全缺失，与原图没有任何匹配。