Mini-project 4

CMPSCI 670, Fall 2019, UMass Amherst
Instructor: Subhransu Maji
Submission by: **Shivam Utreja**

1 Scale-space blob detection [45 points]

1.a Implementation of detectBlobs.py

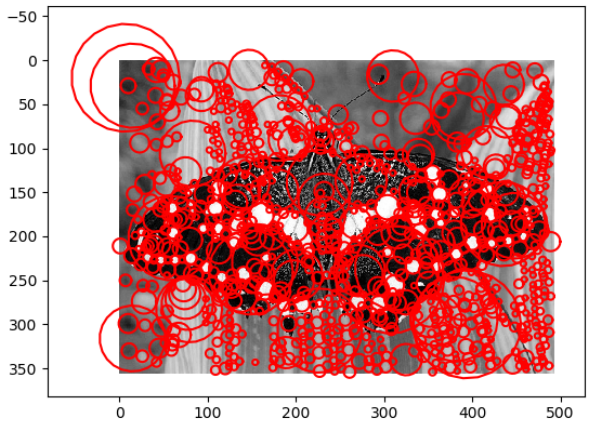
The code for detectBlobs.py is shown below:

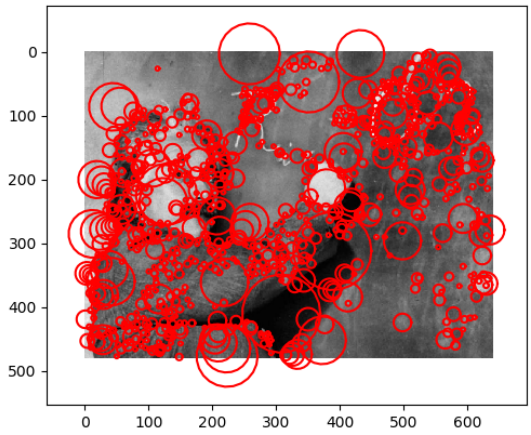
```
# This code is part of:
#
#   CMPSCI 670: Computer Vision
#   University of Massachusetts, Amherst
#   Instructor: Subhransu Maji
#

import numpy as np
import cv2
from skimage.color import rgb2gray
from scipy.ndimage import gaussian_laplace
from scipy.ndimage import generic_filter

def detectBlobs(im, param=None):
    # Input:
    #   IM - input image
    #
    # Ouput:
    #   BLOBS - n x 5 array with blob in each row in (x, y, radius, angle, score)



    # Dummy - returns a blob at the center of the image

    #Preprocessing the image matrix to convert to grayscale
    # and to normalize the values between 0 and 1
    im = rgb2gray(im)

    im = (im - np.amin(im))/(np.amax(im) - np.amin(im))

    # Initial sigma
    s0 = 1.75

    # Factor with which sigma grows
    f = 1.28
```

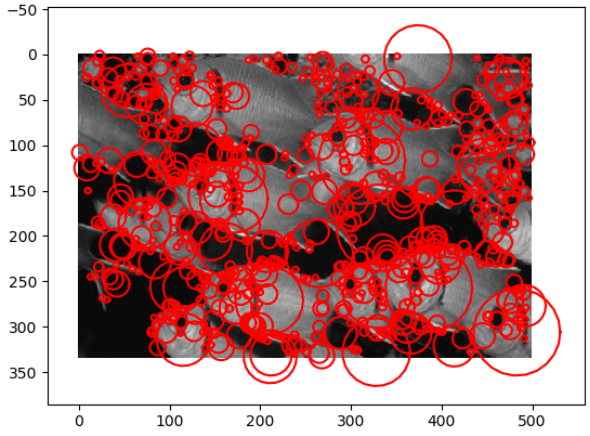
```
# Number of scales of the laplacian of gaussian
n = 15

# Threshold on the value of filter output.
# Values below threshold are ignored.
# threshold = 3e-3
threshold = 0.

# Now we compute the scale space matrix by
# convolving the image with the laplacian of gaussian
# at each of the 15 sigma values
scaleSpace = np.zeros((im.shape[0], im.shape[1], n))

for i in range(n):
    scaleSpace[:,:,i] = gaussian_laplace(im, s0*(f**i), mode='nearest')*((s0*(f**i))**2)

# Squaring the scale space so that we can identify
# both, positive and negative extremes
# which helps us identify both, the white and black blobs.
scaleSpace = scaleSpace**2

blobs = []

# for each entry in the scale space matrix, it is a valid blob
# if it is bigger than all its neighbours.

# this loop performs non-maximum suppression
# and finds the blobs simultaneously
for i in range(1,im.shape[0]-1):
    for j in range(1,im.shape[1]-1):
        for k in range(1,n-1):
            if scaleSpace[i][j][k]==np.amax(scaleSpace[i-1:i+2, j-1:j+2, k-1:k+2]):
                if scaleSpace[i][j][k]>=threshold:
                    blobs.append([j,i,s0*(f**k)*(2**(0.5)), scaleSpace[i][j][k], -1])

print(len(blobs))



#Dummy
# blobs = np.array([[im.shape[1]/2, im.shape[0]/2, 100, 1.0]])

return np.array(blobs)
```

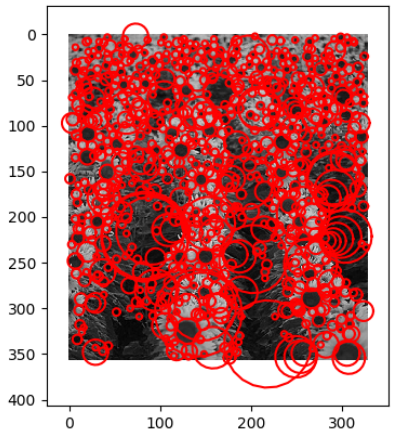
1.b Output of Blob Detection

The output of blob detection for each of the test images is shown below:









2 Image stitching [35 points]

2.a Implementation of computeMatches.py

The code for computeMatches.py is shown below:

```
import numpy as np
from scipy.spatial.distance import cdist

# This code is part of:
#
#   CMPSCI 670: Computer Vision, Fall 2019
#   University of Massachusetts, Amherst
#   Instructor: Subhransu Maji
#
#   Mini-project 4

def computeMatches(f1, f2):
    """ Match two sets of SIFT features f1 and f2 """
    # implement this

    # f1 and f2 are Nx128 and Mx128

    # finding the distances between every pair of vectors from
    # f1 and f2
    distances = cdist(f1,f2, metric='euclidean')

    # sorting the distances to use later
    # in the ratio test
    distancesSorted = np.sort(distances, axis=1)

    # for each entry of f1, finding the closest
    # entry from f2
    matches = np.argmin(distances, axis=1)

    # removing matches if the ratio
    # of the 2 nearest neighbors is more than 0.8
    for i in range(f1.shape[0]):
        if distancesSorted[i][0]>0.8*distancesSorted[i][1]:
            matches[i]=0

    return matches
```

2.b Implementation of ransac.py

The code for ransac.py is shown below:

```
import numpy as np
import cv2
import time

# This code is part of:
#
#   CMPSCI 670: Computer Vision, Fall 2019
#   University of Massachusetts, Amherst
#   Instructor: Subhransu Maji
#
#   Mini-project 4

# chooses a random subset of size n from those matches
# that are not zero
def randomPoints(matches, n):
    nonzero = []

    for i in range(len(matches)):
        if matches[i] != 0:
            nonzero.append(i)

    return np.random.choice(nonzero, size=(n,), replace=False)

def ransac(matches, blobs1, blobs2):

    inliers = []

    transform = None

    # we will choose a random set of 10 points,
    # 200 times
    for i in range(200):

        inl = []

        points = randomPoints(matches, 10)

        A = []
        b = []

        # using coordinates of each of the chosen points
        # to form the linear system of equations
        # to estimate the transform
        for j in points:
            A.append([blobs2[matches[j]][0], blobs2[matches[j]][1], 1., 0., 0., 0.])
            A.append([ 0., 0., 0., blobs2[matches[j]][0], blobs2[matches[j]][1], 1.])
            b.append(blobs1[j][0])
            b.append(blobs1[j][1])

        # the transform is the least squares estimate using the above
        # constructed A and b
```

```
    T = np.linalg.lstsq(A,b, rcond=None)

    # for each pair of blob matches,
    # estimating if that pair is an inlier based on the
    # transform T estimated above
    for k in range(len(matches)):
        if matches[k]!=0:
          xprime= np.dot([blobs2[matches[j]][0],blobs2[matches[j]][1],1.,0.,0.,0.],T[0])
          yprime= np.dot([0.,0.,0.,blobs2[matches[j]][0],blobs2[matches[j]][1],1.],T[0])


          if np.sqrt((blobs1[k][0]-xprime)**2+(blobs1[k][1]-yprime)**2)<1.:
              inl.append(k)

    # if the number of inliers for the current estimate of the transform
    # are more, we update our predicted transformation.
    if len(inl) >= len(inliers):
        inliers = inl
        transform = T[0]

transform = transform.reshape((2,3))

print(transform)

return (inliers, transform)
```

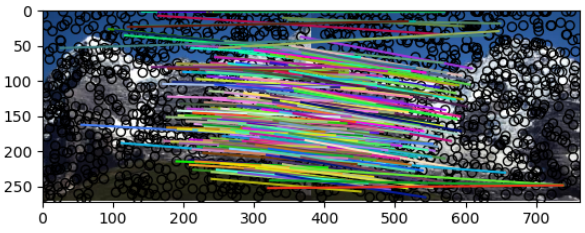
2.c Estimated Affine Transformations

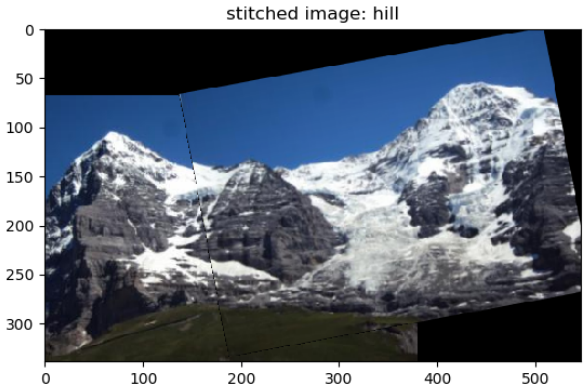
The affine transformations estimated for each of the pairs of images is given below. The transformation sends blobs from image 2 to blobs from image 1.

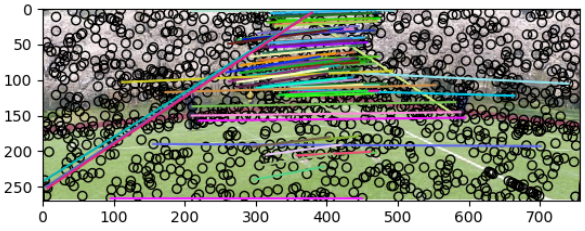
Image	Estimated Transform
hill	[[0.9749576 0.1856631 125.04590335] [-0.17775972 0.98645638 -0.21974834]]
field	[[9.82005811e-01 2.12378666e-01 2.32719706e+02] [-2.13997886e-01 9.95628831e-01 2.45090508e+01]]
ledge	[[9.87001477e-01 9.05426363e-02 1.11159132e+02] [-5.88626416e-02 9.81806036e-01 -1.14805143e+02]]
pier	[[9.86318236e-01 8.28209362e-02 2.76876290e+02] [-8.34714950e-02 9.94005647e-01 3.60436770e+01]]
river	[[9.70749806e-01 -2.32998794e-01 3.00846260e+02] [2.44966141e-01 9.82811493e-01 -6.21416999e+01]]
roofs	[[9.92581274e-01 2.50741337e-01 -1.98815144e+02] [3.25655623e-02 1.08655095e+00 -4.91442468e+01]]
building	[[9.82861189e-01 3.21735644e-02 1.09660073e+02] [7.67777280e-03 9.65066061e-01 1.93869106e+01]]
uttower	[[1.02346096e+00 5.10404674e-02 -2.28236415e+02] [-1.86764605e-02 1.00992408e+00 -1.06695677e+01]]

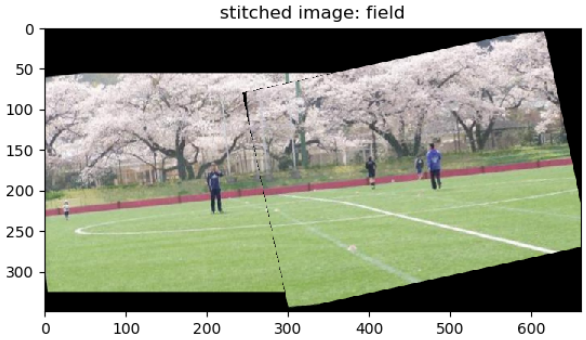
2.d Visualizing the Results

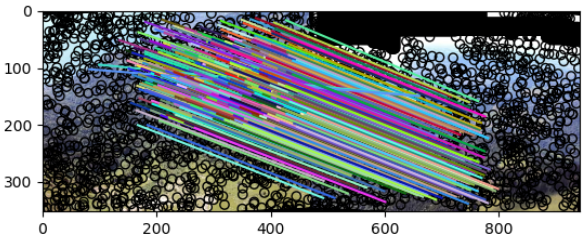
The output of evalStitching.py for each of the images is shown below. The image on the left shows the pairs of matches, and the image on the right shows the stitched image output.

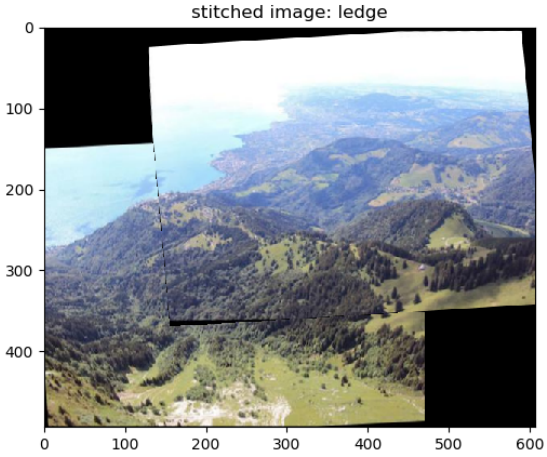


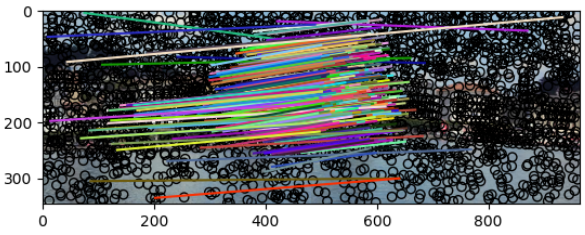


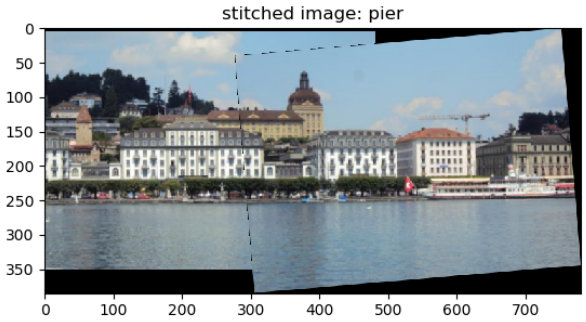


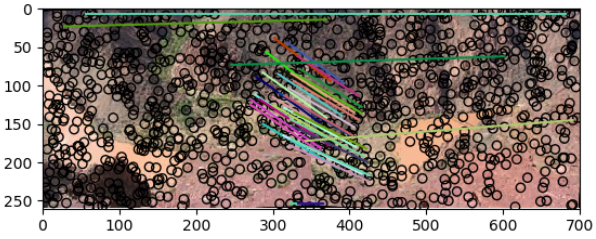


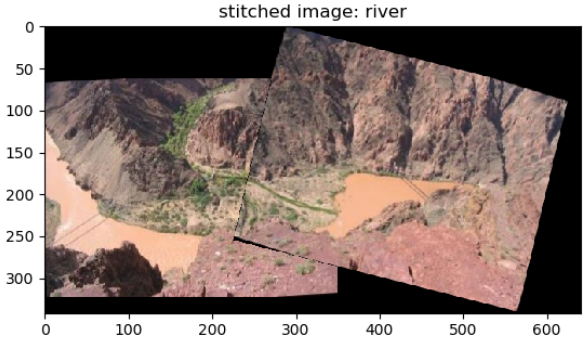


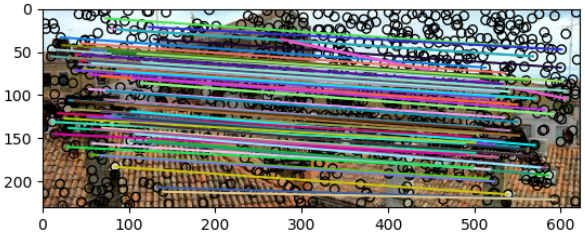


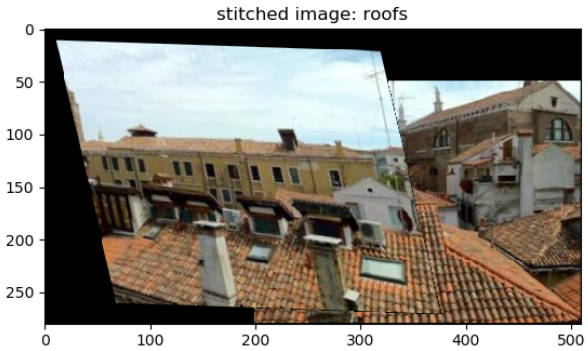


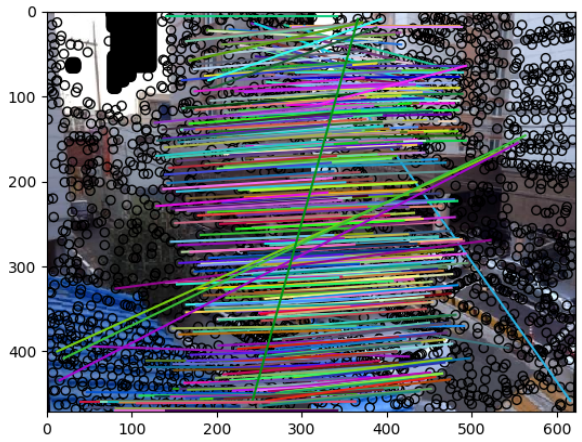

stitched image: pier

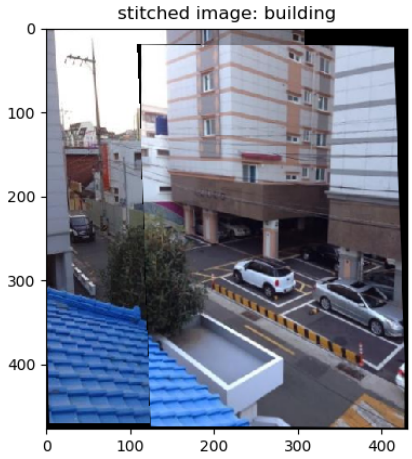


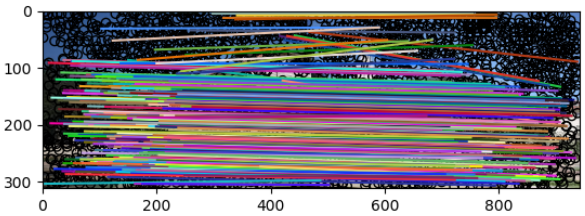

stitched image: river

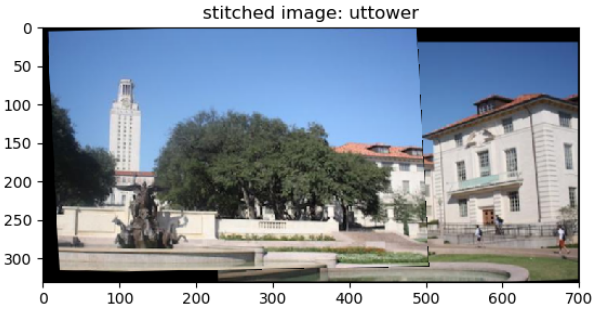



stitched image: roofs


0
100
200
300
400
0
100
200
300
400
500
600


stitched image: building
0
100
200
300
400
0
100
200
300
400


0
100
200
300
0
200
400
600
800


stitched image: uttower
0
50
100
150
200
250
300
0
100
200
300
400
500
600
700