Image Stitching

COMPUTER VISION _ SPRING 2019

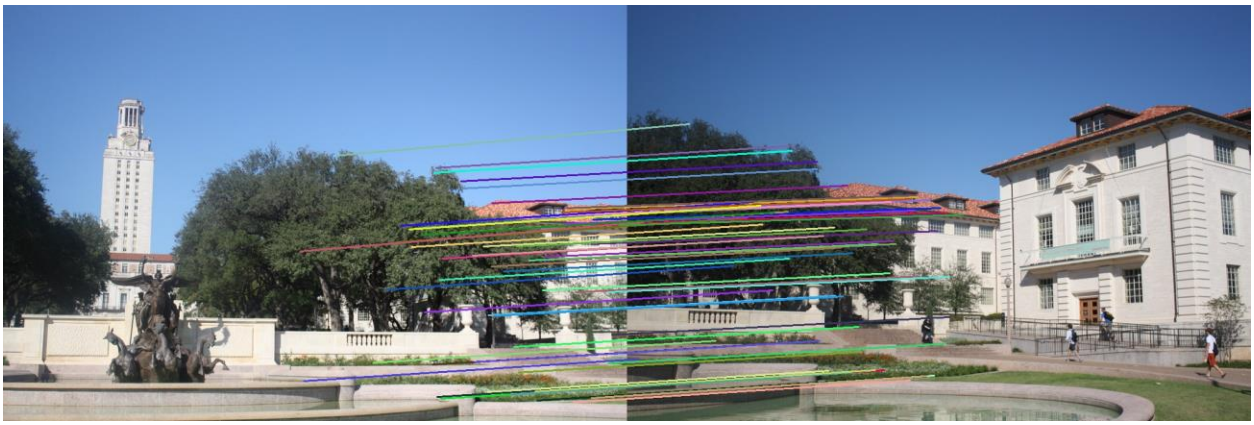
K Anjali Poornima(S20160020132)

Stitching Pairs of Images

Analysing the steps for stitching a pair of images:

1. Consider a pair of images which are to be stitched together.

2. Compute the sift-keypoints and descriptors for both the images using cv2.xfeatures2d.SIFT_create().detectAndCompute().
3. The distances between every descriptor in one image and every descriptor in the other image are computed.
4. All pairs whose descriptor distances are below a specified threshold, or the top few hundred descriptor pairs with the smallest pairwise distances are selected as matched key points.
5. Feature matching is done based on the feature distances computed previous by taking the ratio of the best and second best match.



6. Homography matrix is needed to perform the transformation, and the homography matrix requires at least 4 matches. Run RANSAC to estimate a homography mapping one image onto the other.
7. Use four matches to initialize the homography in each iteration. The output should be a single transformation, H , that gets the most inliers in the course of all the iterations.
8. Warp one image onto the other using the estimated transformation.
9. Create a new image big enough to hold the panorama and composite the two images into it.

10. After aligning the images, the two images are blended seamlessly. Feathering is used to smoothen the line of merging. This is done using cv2.addWeighted(), to give weights for the right and left images and then merge then. In our implementation, we used a window size of 10.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from operator import itemgetter
def getMatchingPoints(kp1, kp2, des1, des2, thr = 0):
        def computeDist(des1, des2):
                return np.linalg.norm(des1 - des2)

        numKp1 = len(kp1)
        numKp2 = len(kp2)
        dist = []
        for i in range(numKp1):
        dTemp = []
        for j in range(numKp2):
        d = computeDist(des1[i, :], des2[j, :])
        dTemp.append((d, i, j))
        sortedA = sorted(dTemp, key=itemgetter(0))[0:2]
        if((thr != 0 and sortedA[0][0] <= thr) or thr == 0):
        dist.append(sortedA)
        if(i%100 == 0):
        print(i, end=" ", flush=True)
        if(thr == 0):
        sDist = sorted(dist, key=lambda x: x[0][0])[0:100]
        return np.array(sDist)
        else:
        return np.array(dist)

def getHomography(P1, P2):

        def getA(X, X1):
                numPt = 4
                A = [] #np.zeros((numPt*2, 9))
                for r in range(0, numPt):
                x, y = X[r] # (u, v)
                u, v = X1[r] # (u', v')
                A.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
                A.append([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
                return A

        ind = np.random.randint(low = 0, high = len(P1), size=4)

        XL = [P1[i].pt for i in ind] # x
```

```
        XR = [P2[i].pt for i in ind] # x'
        A = getA(XL, XR)
        u, s, v = np.linalg.svd(A)
        L = v[-1,:] / v[-1,-1]
        h = L.reshape(3, 3)#np.transpose(np.reshape(v[:, v.shape[1]-1], (3,3)))
        return h
def getNumInliers(H, P1, P2, thr = 5.0):
        def getDist(x, y):
                return np.linalg.norm(x - y)
        inliers = 0
        for i in range(len(mpL)):
        a, b = P1[i].pt
        y = H.dot(np.reshape(np.array([a, b, 1]), (3,1)))
        a1, b1 = P2[i].pt;
        y1 = np.reshape(np.array([a1, b1, 1]), (3, 1))
        if(getDist(y, y1) <= thr):
                inliers += 1
        return inliers
def Blend(thr1, thr2, warp, left, window = 3):
        temp = warp[0:left.shape[0],left.shape[1]:warp.shape[1]]
        temp1 = temp[0:left.shape[0],window:temp.shape[1]]
        temp2 = temp[0:left.shape[0],0:window]
        blend = cv2.addWeighted(left[0:left.shape[0],left.shape[1]-window:left.shape[1]], thr1,temp2, thr1, 0)
        blend1 = cv2.addWeighted(temp1, thr2, temp1, thr2, 0)#0.595
        blend2 = np.concatenate((left[0:left.shape[0],0:left.shape[1]-window], blend), axis=1)
        return np.concatenate((blend2, blend1), axis=1)

left = cv2.imread("left_uttower.JPG")
le = left
leftG = cv2.cvtColor(left, cv2.COLOR_BGR2GRAY)
right = cv2.imread("right_uttower.JPG")
rightG = cv2.cvtColor(right, cv2.COLOR_BGR2GRAY)
ri = right
sift = cv2.xfeatures2d.SIFT_create()
# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(leftG, None)
kp2, des2 = sift.detectAndCompute(rightG, None)
print(np.array(kp1).shape)
matchedPlt = getMatchingPoints(kp1, kp2, des1, des2)
print(matchedPlt, matchedPlt.shape)
good = []
for m1, m2 in matchedPlt:
        if(m1[0] < 0.3*m2[0]):
                good.append(m1)
good = np.array(good)
print(good.shape)
mpL = []
mpR = []
imgL = left.copy()
imgR = right.copy()
for _, i, j in good:
        mpL.append(kp1[int(i)])
        mpR.append(kp2[int(j)])
        a, b = kp1[int(i)].pt
        cv2.circle(imgL, (int(a), int(b)), 2, 255, -1)
        a, b = kp2[int(j)].pt
        cv2.circle(imgR, (int(a), int(b)), 2, 255, -1)
print(np.array(mpL).shape, np.array(mpR).shape)
cv2.imwrite("LeftMatchingPoints.jpg", imgL)
cv2.imwrite("RightMatchingPoints.jpg", imgR)
hA, wA = left.shape[:2]
hB, wB = right.shape[:2]

vis = np.zeros((max(hA, hB), wA + wB, 3), dtype="uint8")
vis[0:hA, 0:wA] = left
vis[0:hB, wA:] = right

for i in range(len(mpL)):
```

```
        ptA = (int(mpL[i].pt[0]), int(mpL[i].pt[1]))
        ptB = (int(mpR[i].pt[0]) + wA, int(mpR[i].pt[1]))
        cv2.line(vis, ptA, ptB, (np.random.randint(0,255), np.random.randint(0,255), np.random.randint(0,255)), 2)
cv2.imwrite("Matchings.jpg", vis)
N = 10
thr = 150.0
H = []
initialIn = 0
for itr in range(N):
        HTemp = getHomography(mpR, mpL)
        inliers = getNumInliers(HTemp, mpR, mpL, thr)
        #print(inliers)
        if(initialIn < inliers):
        initialIn = inliers
        H = HTemp
print(H)
warp = cv2.warpPerspective(right, H, (right.shape[1] + left.shape[1], right.shape[0]))
plt.imshow(warp),plt.title('Warped Image')
plt.show()
plt.figure()
warp[0:left.shape[0], 0:left.shape[1]] = left
cv2.imwrite('resultant_stitched_panorama.jpg',warp)
plt.imshow(warp)
plt.show()
final = Blend(0.55, 0.595, warp, left, 10)
plt.imshow(final)
cv2.imwrite('Result_afterBlending.jpg',final)
```

Observations:

As the threshold for inliers increases keeping the same number of iterations, the number of inliers increases and average residual increases.

As we increase the number of iterations, keeping the threshold constant, the inliers increase and the average residual decreases.

# Iterations	Threshold	# Inliers	Average Residual	#Good Points
10	10	7	6.791	67(First Hundred)
10	20	27	12.855	67
10	100	61	40.26	67
100	10	15	4.204	67
100	40	66	15.7727	67
10	10	44	5.688	248(511 matched points)
100	25	226	11.8	248

The observation is that, the resultant image is not aligned properly if the inliers are small, since the homography matrix cannot align all the points in the other image properly.

We have also tried with different thresholds while calculating the matched points from sift features. For example : threshold 50 - 27 matched points, threshold 100 - 511 matched points, threshold 75 - 191 matched points