CSE-578 Computer Vision

Assignment 2
Niharika Vadlamudi
2018122008

March 1 2020

1 Image Mosiacing

The goal is to use any feature detector and descriptor (e.g. SIFT) to find matches between two partially overlapping images.Estimate the Homography matrix between the two images robustly.Transform one of the images to the others reference frame using the Homography matrix.Stitch the two/multiple images together to produce a singly mosaic/panorama.

In this part , we use SIFT features to generate key points , we can use Harris Features - ORB features .

1.1 SIFT

```
def siftfeatures(img):
    sift=cv2.xfeatures2d.SIFT_create()
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    #half = cv2.resize(gray,(0,0), fx = 0.2, fy = 0.2)
    kp, des = sift.detectAndCompute(gray,None)
    return kp,des
```

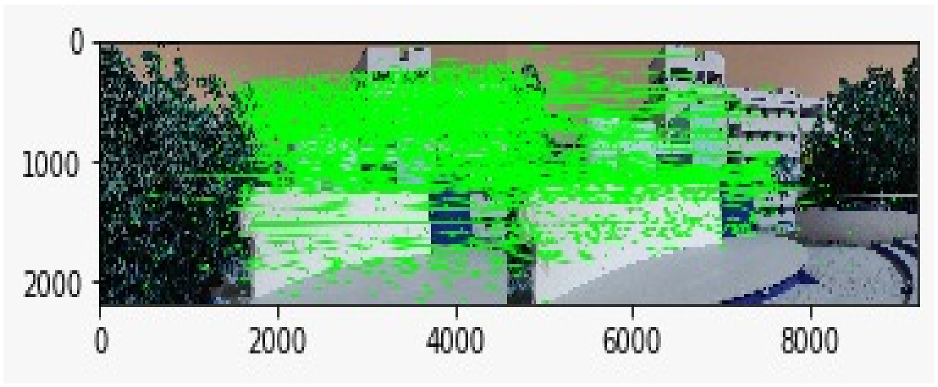

Figure 1: Original Image

Now,matching of these features can be done via many methods , if we want fast detection bf.knnMatching() and bf.FlannDetecting().

1.2 Matching the features

```
def siftfeatures(img):
    sift=cv2.xfeatures2d.SIFT_create()
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    #half = cv2.resize(gray,(0,0), fx = 0.2, fy = 0.2)
    kp, des = sift.detectAndCompute(gray,None)
    return kp,des
```

1.3 Homography Matrix

We need atleast 4 points for Homography matrix computation . ALso , like DLT we also take 4 random points and compute the optimal H matrix with least normalised mean square error .

```
def computeRow(pointP,pointQ):
    row1=[-pointP[0],-pointP[1],-1,0,0,0,pointP[0]*pointQ[0],pointP[1]*pointQ[0],pointQ[0]]
    row2=[0,0,0,-pointP[0],-pointP[1],-1,pointP[0]*pointQ[1],pointP[1]*pointQ[1],pointQ[1]]

    return(np.vstack((row1,row2)))

def homographyMatrix(srcpts,despts):
    M=[]
    N=len(srcpts);
    for i in zip(srcpts,despts):
        A=computeRow(i[0],i[1])
        M.append(A)
    M=np.asarray(M)
    m,n,p=M.shape
    M=np.reshape(np.asarray(M),(m*n,p))

    # Computing the SVD part .
    U, D, V_T =np.linalg.svd(M)
    #Extrating the last column of V
    H=np.reshape(V_T[-1,:]/V_T[-1,-1],(3,3))

    # MSE Error .
    # Calculating the Mean error now , as H is known to us .
    # Take up all world co-ordinates ,mul by H , we know the corresponding image co-ordinates .

    mse=0;
    despt_append=[ [despts[i][0],despts[i][1],1] for i in range(0,len(despts)) ]

    for i in range(0,1,len(despts)) :
        xcPred=np.dot(H,np.reshape(np.asarray(despt_append[i]),(3,1)));
        xcPred=xcPred[0:2]/xcPred[-1]
        mse=mse+np.linalg.norm((np.reshape(np.asarray(despts[i]),(2,1))-xcPred))

    mse=mse/len(despts)
    return(H,mse)
```

1.4 RANSAC:Homography

Now,matching of these features can be done via many methods , if we want fast detection bf.knnMatching() and bf.FlannDetecting().

```
def homographyRANSAC(scrpts,despts,verbose=False):
    #For Computation  of Homography any 4 pts are enough.
    scrpts=list(scrpts)
    despts=list(despts)
    N=len(scrpts)

    if(N<4):
        print('Homography cannot be computed , pls enter more points')
        return(0);

    mse_max=1000000;
    for subset in itertools.combinations(range(0,N),4):
        Si=[scrpts[i] for i in subset]
        Di=[despts[i] for i in subset]
        # Calling Homography Function .
        Hi,erri=homographyMatrix(Si,Di)
        if(erri<mse_max):
            H,mse=homographyMatrix(Si,Di)
            msemax=mse
    return(H,mse)
```

1.5 Results-Homography Matrix

```
H= [[ 1.69197263e+00 -1.63722968e-03 -2.54683749e+03]
 [ 1.69218993e-01  1.44760102e+00 -5.61388706e+02]
 [ 1.49700749e-04  7.52003600e-06  1.00000000e+00]]
MSE=38.56
```

Now,matching of these features can be done via many methods , if we want fast detection bf.knnMatching() and bf.FlannDetecting().

1.6 Warping Image

```
def warp_project(imageA, imageB, homography,verbose=True):
    h1, w1 = imageA.shape[:2]
    h2, w2 = imageB.shape[:2]
    pts1 = np.float32([[0, 0], [0, h1], [w1, h1], [w1, 0]]).reshape(-1, 1, 2)
    pts2 = np.float32([[0, 0], [0, h2], [w2, h2], [w2, 0]]).reshape(-1, 1, 2)

    pts2_ = cv2.perspectiveTransform(pts2, homography)
    pts = np.concatenate((pts1, pts2_), axis=0)

    [xmin, ymin] = np.int8(pts.min(axis=0).ravel() - 0.5)
    [xmax, ymax] = np.int8(pts.max(axis=0).ravel() + 0.5)
    t = [-xmin, -ymin]
    Ht = np.array([[1, 0, t[0]], [0, 1, t[1]], [0, 0, 1]])
    result = cv2.warpPerspective(imageB, Ht.dot(homography), (xmax-xmin, ymax-ymin))
    cv2.imwrite("project.png", result)
    result[t[1]:h1+t[1], t[0]:w1+t[0]] = imageA

    if(verbose):
        fig=plt.figure(figsize=[10,10])
        gray = cv2.cvtColor(result, cv2.COLOR_RGB2GRAY)
        plt.imshow(gray)
        plt.title('Stiched Image')
    return(result)
```

2 Panorama Results

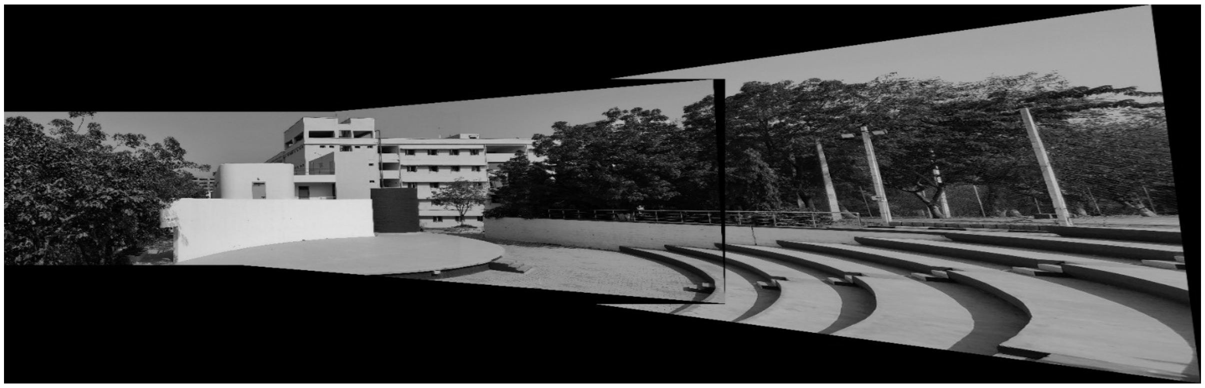

Figure 2: Case 1

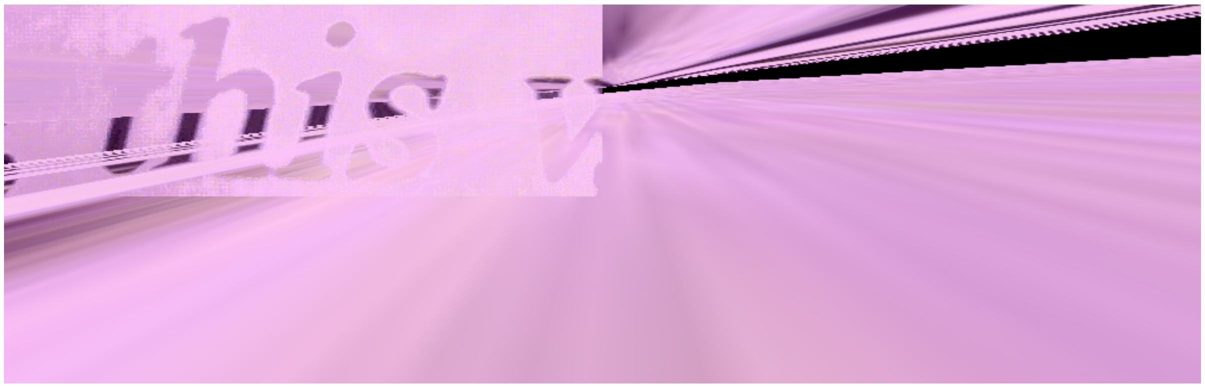

Figure 3: Case 1

3 Stereo Correspondences

We firstly perform intensity patch based matching which is a crud substitute for SIFT/SURF matching . Here, we try to search 1 patch across the entire image .

3.1 Window Intensity Matching

```python

 # Define Correlation as dot product(normalized)
def corr(v1,v2):
    return v1.T.dot(v2)/(np.sqrt(v1.T.dot(v1))*np.sqrt(v2.T.dot(v2)))
def correlation_matching(img1,img2,window_size=128,stride=128,thresh = 0.01):
    h1,w1,c = img1.shape
    h2,w2,c = img2.shape

#     Pass through all the patches in img1 and find patch in img2 with least
    best_matches = []
    for y1 in range(0,h1-window_size,stride):
        for x1 in range(0,w1-window_size,stride):
            least_dis = 1.0
            least_coord = []
            for y2 in range(0,h2-window_size,stride):
                for x2 in range(0,w2-window_size,stride):
                    v1 = img1[y1:y1+window_size, x1:x1+window_size,:].flatten()
                    v2 = img2[y2:y2+window_size, x2:x2+window_size,:].flatten()
                    dis = corr(v1,v2)
                    if least_dis > dis:
                        least_dis = dis
                        least_coord = [x1,y1,x2,y2,dis]
            best_matches.append(least_coord)
    return best_matches

def draw_matches(img,matches,window_size=128):
    h,w,c = img.shape
#     print(len(matches))
    line_img = []
    for match in matches:
        while len(match) < 4:
            match.append(0)
#         print(len(match))
        pt1 = (match[1]+window_size//2,match[0]+window_size//2)
        pt2 = (match[3]+window_size//2+w//2,match[2]+ window_size//2)
        line_img = cv2.line(img,pt1,pt2,(0,0,225),3)

    return line_img
```

3.2 Results

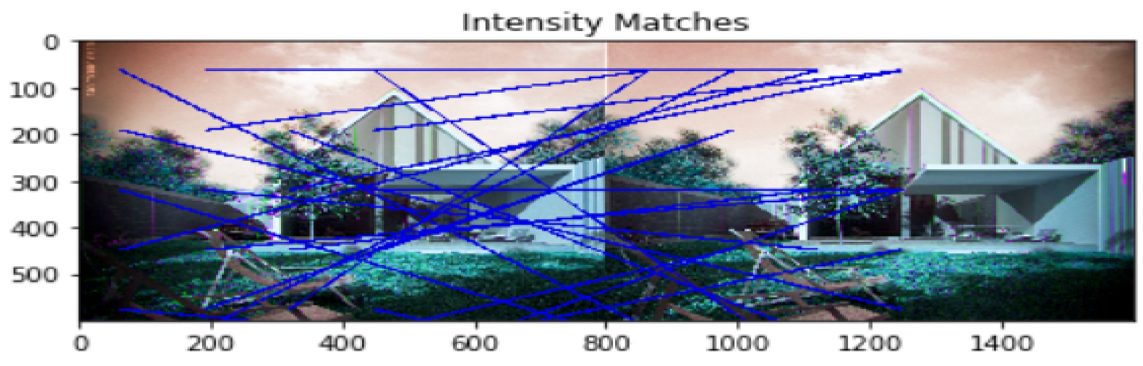

Figure 4: Case 1

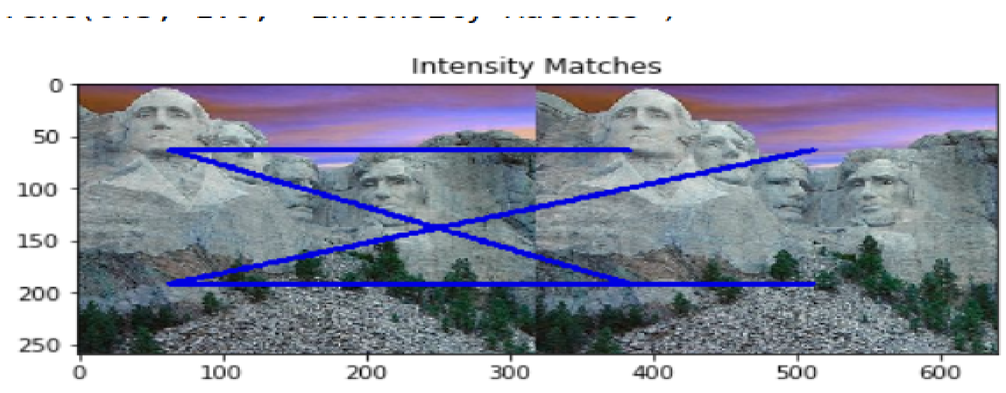

Figure 5: Case 1

3.3 Stereo Rectification

```python
def stereoRectification(im1,im2,match_thresh=0.8,verbose=False):
    #Generate SIFT parameters & store the matches .
    matches,kp1,kp2,pts1,pts2=featurematchingknn(img1,img2,verbose=False)
    #RANSAC inbuilt function .
    F,mask = cv2.findFundamentalMat(pts1,pts2,cv2.RANSAC)
    # We select only inlier points
    pts1 = pts1[mask.ravel()==1]
    pts2 = pts2[mask.ravel()==1]
    img_size = img1.shape[0:2
    p,H1,H2=cv2.stereoRectifyUncalibrated(pts1, pts2, F, img_size)
    H3=H1.dot(H2)
    img1_corrected = cv2.warpPerspective(img1, H1, img_size)
    img2_corrected = cv2.warpPerspective(img2, H3, img_size)

    imgadd=np.hstack((np.asarray(img2_corrected),np.asarray(img1_corrected)))
    return imgadd,img1_corrected, img2_corrected
```

3.4 Results: Window Based Matching after Stereo Rectification

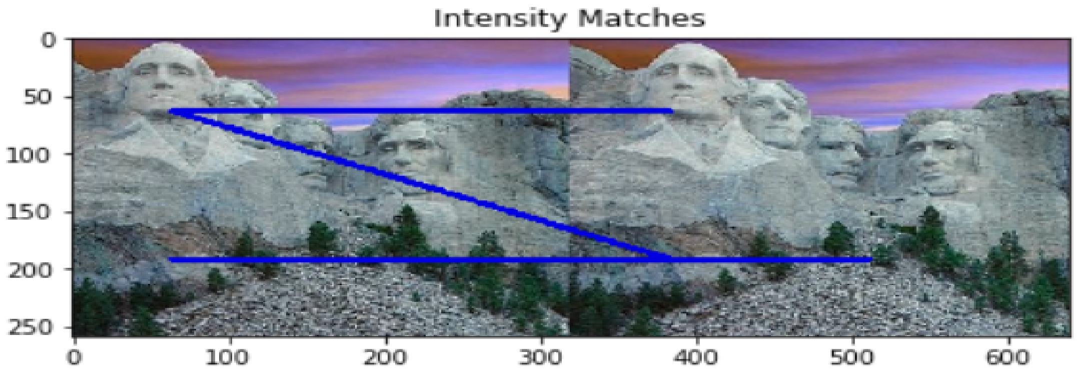

Figure 6: Case 1

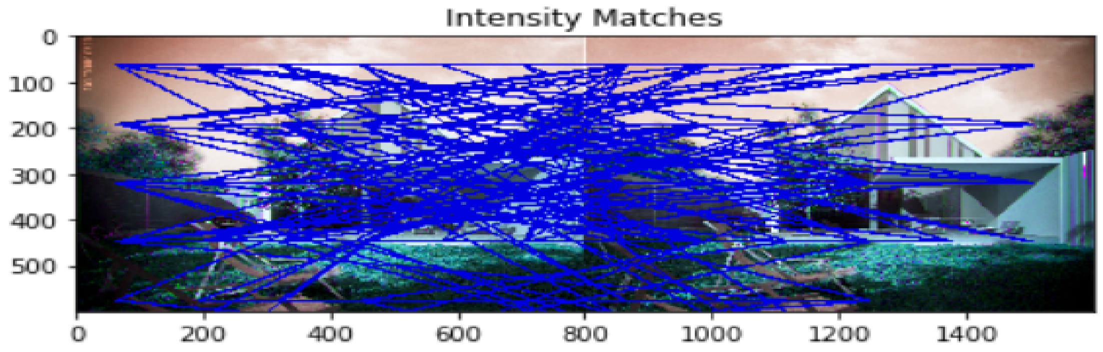

Figure 7: Case 1

4 Conclusion

I got deeper perspective of Image Mosiacing and techniques , also complete implementation of DLT made me understand the mathematical flair behind the process.A lot of new OpenCV libraries and functions were learnt in teh process.