**EN2550 Assignment 2 on Fitting and Alignment**

**Index: 190018V** **Name: Abeywickrama K.C.S.**

GitHub Link: <https://github.com/KCSAbeywickrama/EN2550-Excercises/tree/master/Assignment_02>

Only important code parts & results have been included. Full code with all function implementation is on [GitHub](https://github.com/KCSAbeywickrama/EN2550-Excercises/tree/master/Assignment_02).

**1) (a)**

def get\_circle(points,max\_r):  # find circle from 3 points

    …

def fit\_circle(points):  # find the best circle from a set of points algebraically

    …

def get\_inliers(points,thres): # find inliers using RANSAC

    point\_count=len(points)

    max\_inlier\_count=0

    max\_r=(np.max(points)-np.min(points))/2

    for ittr in range(ittrs\_limit):

        init\_points=points[np.random.choice(point\_count,3)]

        circle=get\_circle(init\_points,max\_r)

        if(circle):

            center,r=circle

            tmp\_diff\_sqr=(points-np.array(center))\*\*2

            r\_difs=np.abs(np.sqrt(tmp\_diff\_sqr[:,0]+tmp\_diff\_sqr[:,1])-r)

            inliers = points[r\_difs<thres]

            inlier\_count=len(inliers)

            if inlier\_count>max\_inlier\_count:

                match\_circle,match\_samples,match\_inliers=circle,init\_points,inliers

                max\_inlier\_count=inlier\_count

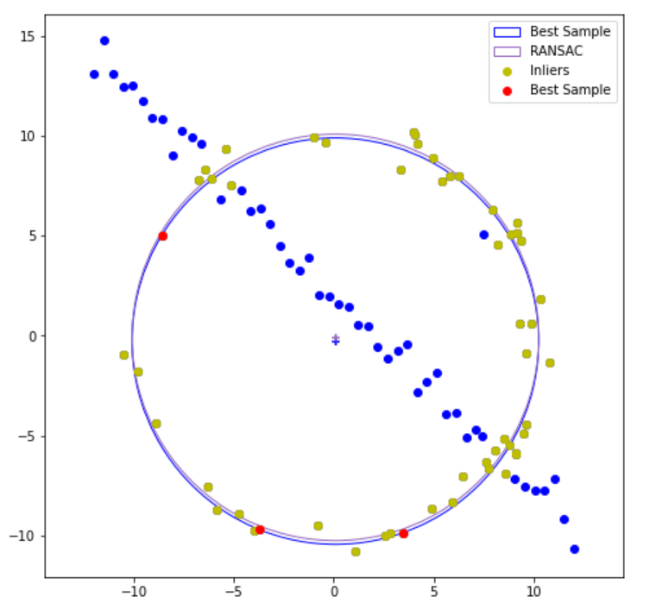
    return match\_circle,match\_samples,match\_inliers

ittrs\_limit=500

inlie\_thres=1

(smpl\_center,smpl\_r),smpl\_points,inliers=get\_inliers(X,inlie\_thres) #get inliers using RANSAC

ransac\_center,ransac\_r=fit\_circle(inliers) #find best circle from inliers using algebriacally



After getting a set of inliers, to get a better result the circle has recalculated by considering all the inliers

**1) (b)**

There is a little difference between the circle from the best matching sample set & the recalculated circle by considering all inliers. (Clear large image is available in the [notebook](https://github.com/KCSAbeywickrama/EN2550-Excercises/blob/master/Assignment_02/190018V_a02.ipynb) on [GitHub](https://github.com/KCSAbeywickrama/EN2550-Excercises/tree/master/Assignment_02))

**2)**

img\_dst=cv.imread('imgs2/00x.jpg')

img\_src=cv.imread('imgs2/flagx.png')

points\_des=np.array(get\_points(img\_dst))

img\_src\_h=img\_src.shape[0]

img\_src\_w=img\_src.shape[1]

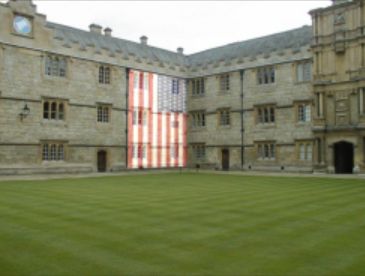
points\_src=np.array([(0,0),(img\_src\_w,0),(img\_src\_w,img\_src\_h),(0,img\_src\_h)])

h,st=cv.findHomography(points\_src,points\_des)

img\_warp=cv.warpPerspective(img\_src,h,img\_dst.shape[1::-1])

img\_out=cv.addWeighted(img\_warp,0.5,img\_dst,1,0.0)

get\_points() is a function that uses GUI options in OpenCV to find destination points of the flag image in the architectural image with mouse clicks (implementation available in the notebook on GitHub). Computing homography, warping & superimposing has done using inbuilt functions in OpenCV.

Images with flat surfaces (large walls) have been selected as architectural images. Since flags are flat superimposing these into flat surfaces on the architectural images ends up with a nice result.

**3) (a)**

img1=cv.imread('imgs3/img1.ppm')

img2=cv.imread('imgs3/img4.ppm')

sift=cv.SIFT\_create()

kp1,des1=sift.detectAndCompute(img1,None)

kp2,des2=sift.detectAndCompute(img2,None)

bf=cv.BFMatcher()

matches=bf.knnMatch(des1,des2,k=2)

good=[];pts1=[];pts2=[]

for m,n in matches:

    if m.distance<0.65\*n.distance:

        pts1.append(kp1[m.queryIdx].pt);pts2.append(kp2[m.trainIdx].pt);good.append([m])

img\_match=cv.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=2)

A wall covered in graffiti

Description automatically generated with low confidenceSince the projective angle between img1 & img5 is very high it is very difficult to find a reasonable amount of matching features using SIFT. Because SIFT does not perform well in large changes in viewpoint angle. So img1 & img4 have been selected to continue the question workout. (workout of trying to match SIFT features between img1 & img5 has been included in GitHub for the completion. But by limiting points to best points by reducing the threshold and then manually checking it can be seen that those are incorrect matches)

**3) (b)**

def compute\_H(pts1,pts2):

    A=[]

    for i in range(len(pts1)):

        xs,ys=pts1[i]

        xd,yd=pts2[i]

        A.append((xs,ys,1,0,0,0,-xd\*xs,-xd\*ys,-xd))

        A.append((0,0,0,xs,ys,1,-yd\*xs,-yd\*ys,-yd))

    A=np.array(A)

    L,V=np.linalg.eig(A.T @ A)

    l=np.argmin(np.abs(L))

    v=V[:,l]

    h33=v[-1]

    return v.reshape((3,3))/h33

max\_inlier\_count=0

for n in range(5000):

    smpl\_indxs=np.random.choice(len(pts1),4,replace=False)

    smpl\_pts1=pts1[smpl\_indxs]

    smpl\_pts2=pts2[smpl\_indxs]

    H=compute\_H(smpl\_pts1,smpl\_pts2)

    Xs=np.vstack((pts1.T,np.ones(pts1.shape[0],dtype=int)))

    Xd=pts2.T

    XdH=H @ Xs

    XdH=XdH/XdH[2]

    XdH=np.delete(XdH,2,axis=0)

    tmp\_diff\_sqr=(XdH-Xd)\*\*2

    dis\_diff=np.sqrt(tmp\_diff\_sqr[0]+tmp\_diff\_sqr[1])

    thres=2

    inlier\_idxs=dis\_diff<thres

    inliers=pts1[inlier\_idxs],pts2[inlier\_idxs]

    inlier\_count=len(inliers[0])

    if(inlier\_count>max\_inlier\_count):

        match\_inliers=inliers

        max\_inlier\_count=inlier\_count

H=compute\_H(\*match\_inliers)

H=

[[ 6.56816848e-01 6.80640096e-01 -3.08528938e+01]

[-1.51582999e-01 9.69920508e-01 1.49997446e+02]

[ 4.09138545e-04 -1.06392211e-05 1.00000000e+00]]

the square root of sum of squared differences between calculated H & given H in the data set is 1.28

So calculated H and given H are almost equal. So H calculation is very accurate. ( Reading given H and the calculating difference is available in the [notebook](https://github.com/KCSAbeywickrama/EN2550-Excercises/blob/master/Assignment_02/190018V_a02.ipynb) on [GitHub](https://github.com/KCSAbeywickrama/EN2550-Excercises/tree/master/Assignment_02))

**3) (c)**

T=np.array([[1,0,50],[0,1,50],[0,0,1]],dtype=float)

img1\_mask=np.ones(img1.shape)

canvas1=cv.warpPerspective(img1,T @ H,(900, 850))

canvas1\_mask=cv.warpPerspective(img1\_mask,T @ H,(900, 850))==1

canvas2=cv.warpPerspective(img2,T,(900, 850))

canvas\_out=np.array(canvas2)

canvas\_out[canvas1\_mask]=canvas1[canvas1\_mask]

A picture containing diagram

Description automatically generatedA picture containing text, different, colorful

Description automatically generatedA picture containing diagram

Description automatically generated

Translation(using T matrix) has been given to img1 & img4 for better results without cropping parts after stitching.

Translation & projection has given to img1 at once by multiplying the Translation matrix and Homography