Image processing Assignment II **190285J Prabhashwara Kahawalage**

GitHub: https://github.com/r3ki3g/Image-processing-fundamentals/tree/main/assignment-2

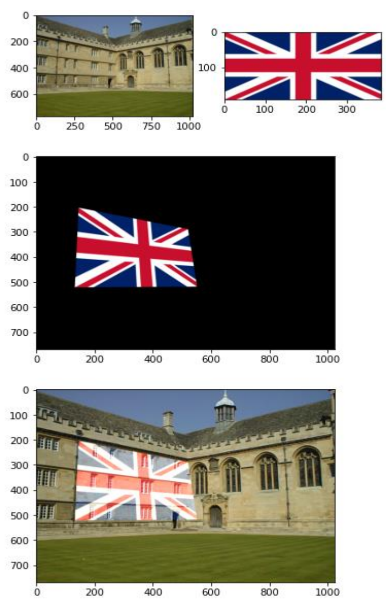
(All the codes and full python notebook is in the GitHub/)

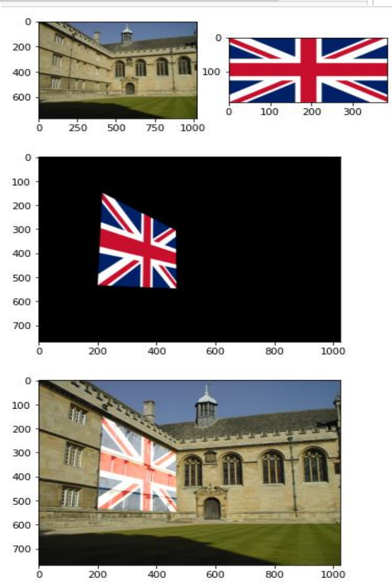
Question 02

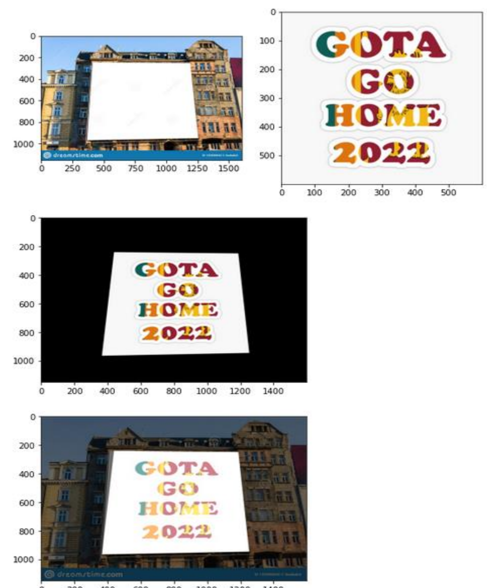
```python
wadham = r"D:/ENTC/SEM_4/EN2550 - Fundamentals of Image Processing and Machine Vision/~images/ass2/wadham/001.jpg"
flag = r"D:/ENTC/SEM_4/EN2550 - Fundamentals of Image Processing and Machine Vision/~images/ass2/Flag_of_the_United_Kingdon

imgWadham = cv.cvtColor(cv.imread(wadham,1),cv.COLOR_BGR2RGB)
imgFlag = cv.cvtColor(cv.imread(flag,1),cv.COLOR_BGR2RGB)

fig, ax = plt.subplots(1,2)
ax[0].imshow(imgWadham)
ax[1].imshow(imgFlag)
plt.show()
X1 = np.array([[0,0],
               [0,192],
               [383,0],
               [383,192]])

X2 = np.array([[146,205],
             [133,522],
             [521,290],
             [552,519]])

H, mask = cv.findHomography(X1, X2, cv.RANSAC, 5.0)

w=1024
h=768

warped = cv.warpPerspective(imgFlag, H, (w, h))

plt.imshow(warped)
plt.show()

superImposed = cv.addWeighted(imgWadham,1,warped,0.7,1)

plt.imshow(superImposed)
plt.show()
```

Important points:

1. "Mouse click" and getting the coordinates was not possible because my notebook (Jupytor) froze every time I tried to do so. **So I had to hardcode the coordinates**.
2. First hardcoded the 4 corner coordinates of the flag then the 4 corner coordinates of the spot we need to super impose.
3. Then calculated the matrix using the cv.findHormography method and warped the flag as needed. Finally super imposed with fine weights using cv.addWeighted.



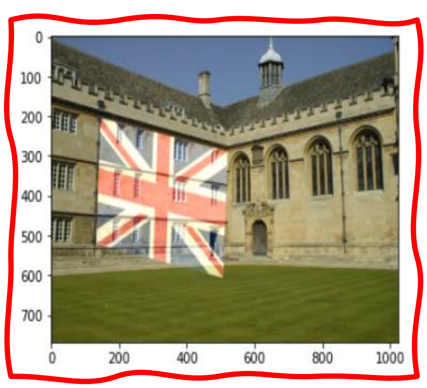




Experimented with finding the hormographic matrix myself

```python
In [43]:
X2 = np.array([[146],
               [133],
               [521]])
Y2 = np.array([[205],
               [522],
               [290]])
Ones2 = np.array([[1],
                  [1],
                  [1]])
M1 = np.array([[0,0,1],
               [0,192,1],
               [383,0,1]])

M1_inv = np.linalg.inv(M1)

Hr1 = M1_inv @ X2
Hr2 = M1_inv @ Y2
Hr3 = M1_inv @ Ones2
H = np.array([Hr1,Hr2,Hr3])

print(H)
w=1024
h=768
warped = cv.warpPerspective(imgFlag, H, (w, h))
superImposed = cv.addWeighted(imgWadham,1,warped,0.5,1)
plt.imshow(superImposed)
plt.show()
```

executed in 276ms, finished 14:11:50 2022-05-11



However what I got was the affine transformation with [0 0 1] as the bottom row. Therefore the warping did not fit the wall.

```
[[[ 9.79112272e-01]
  [-6.77083333e-02]
  [ 1.46000000e+02]]

 [[ 2.21932115e-01]
  [ 1.65104167e+00]
  [ 2.05000000e+02]]

 [[ 0.00000000e+00]
  [ 0.00000000e+00]
  [ 1.00000000e+00]]]
```

Question 03

```python
def siftMatchFinder(img1,img2):
    sift = cv.SIFT_create()
    im1, des_1 = sift.detectAndCompute(img1, None)
    im2, des_2 = sift.detectAndCompute(img2, None)
    T1 = cv.BFMatcher(cv.NORM_L1, crossCheck = True)
    Match = T1.match(des_1, des_2)
    sortMatch = sorted(Match, key = lambda x:x.distance)
    img4 = cv.drawMatches(img1, im1, img2, im2, sortMatch[:100], img2, flags =2)
    fig, ax = plt.subplots(1,1, figsize = (18, 18))
    ax.imshow(img4)
    ax.axis('off')
    plt.show()
    return Match,[im1,im2]
```

It was impossible to find matches between 1.ppm and 5.ppm directly. So idea was to find the hormography between 1.ppm and 2.ppm then 2.ppm and 3.ppm , and like so finally 4.ppm and 5.ppm. Then we can pre-multiply the hormographies and get the composition of hormography which is equivalent to hormography between 1.ppm and 5.ppm

siftMatchFinder() function is defined to calculates sift matches between images. (It is used multiple times later in the code)

```python
or corr in corres:
    pts1 = np.matrix([corr.item(0), corr.item(1), 1])
    pts2 = np.matrix([corr.item(2), corr.item(3), 1])

    A2 = [0, 0, 0, -pts2.item(2) * pts1.item(0), -pts2.item(2) * pts1.item(1), -pts2.item(2) * pts1.item(2),pts2.item(
         pts2.item(1) * pts1.item(1), pts2.item(1) * pts1.item(2)]
    A1 = [-pts2.item(2) * pts1.item(0), -pts2.item(2) * pts1.item(1), -pts2.item(2) * pts1.item(2), 0, 0, 0, pts2.item
         pts2.item(0) * pts1.item(1), pts2.item(0) * pts1.item(2)]
```

Creating the data matrix to find the least error transform with RANSAC algorithm. Using matrix and Numpy library helps do the calculations fast and ends up letting us do much more iterations for the RANSAC.

```python
def getDistanceBtwMat(corres, H):
    pts1 = np.transpose(np.matrix([corres[0].item(0), corres[0].item(1), 1]))
    es_pts1 = np.dot(H, pts1)
    es_pts2 = (1/es_pts1.item(2))*es_pts1

    pts2 = np.transpose(np.matrix([corres[0].item(2), corres[0].item(3), 1]))
    err = pts2 - es_pts2
    return np.linalg.norm(err)

def HormographyFromRANSAC(corres, thresh):
    final = None
    max_inliers = []
    for i in range(1000):
        corres1 = corres[random.randrange(0, len(corres))]
        corres2 = corres[random.randrange(0, len(corres))]
        rand_four = np.vstack((corres1, corres2))
        corres3 = corres[random.randrange(0, len(corres))]
        rand_four = np.vstack((rand_four, corres3))
        corres4 = corres[random.randrange(0, len(corres))]
        rand_four = np.vstack((rand_four, corres4))
        H = calc_Homography(rand_four)
        inliers = []
        for i in range(len(corres)):
            d = getDistanceBtwMat(corres[i], H)
            if d < 5:
                inliers.append(corres[i])
        if len(inliers) > len(max_inliers):
            max_inliers = inliers
            final = H
        if len(max_inliers) > (len(corres)*thresh):
            break
    return final, max_inliers
```

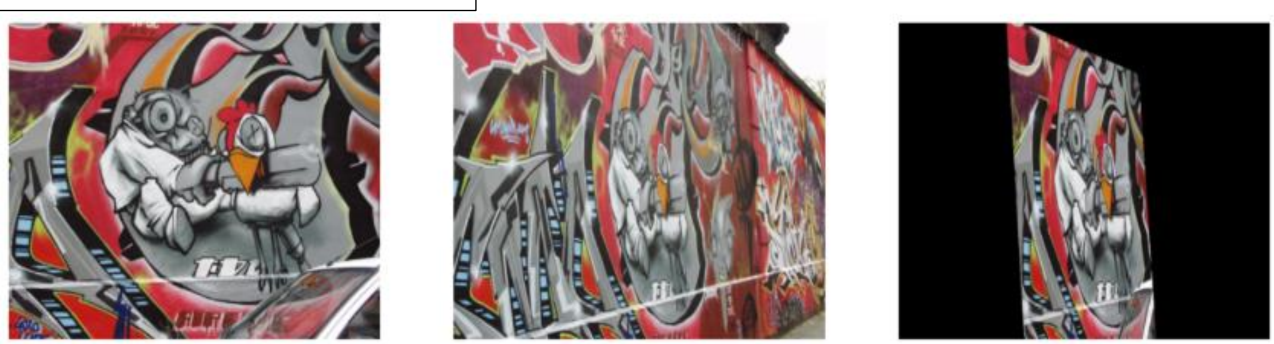
```python
M1,K1=siftMatchFinder(img1,img2)
corres_lis1=corr_list(M1,K1)
corres1 = np.matrix(corres_lis1)
H1, inliers1 = HormographyFromRANSAC(corres1, 0.6)

M2,K2=siftMatchFinder(img2,img3)
corres_lis2=corr_list(M2,K2)
corres2 = np.matrix(corres_lis2)
H2, inliers2 = HormographyFromRANSAC(corres2, 0.6)

M3,K3=siftMatchFinder(img3,img4)
corres_lis3=corr_list(M3,K3)
corres3 = np.matrix(corres_lis3)
H3, inliers3 = HormographyFromRANSAC(corres3, 0.6)

M4,K4=siftMatchFinder(img4,img5)
corres_lis4=corr_list(M4,K4)
corres4 = np.matrix(corres_lis4)
H4, inliers4 = HormographyFromRANSAC(corres4, 0.6)

H = H4 @ H3 @ H2 @ H1
print("hormography compositioned" ,H)

DST1 = cv.warpPerspective(img1, H, ((img5.shape[1]), img5.shape[0]))
```

getDistanceBtwMat() function calculated the geometric distance between given vectors. It is used to see how the destination points are deviated from the requires positions in the image.

By selecting the case of minimal eigen values, in each iteration we can calculate the hormography matrix . The result is shown here:

```
hormography compositioned [[ 5.65738181e-01  5.79922504e-02  2.30593711e+02]
 [ 1.91483499e-01  1.13184499e+00 -1.56137084e+01]
 [ 4.04731077e-04 -4.21852199e-05  9.97146955e-01]]
```

Final stitched image and warped version are shown here

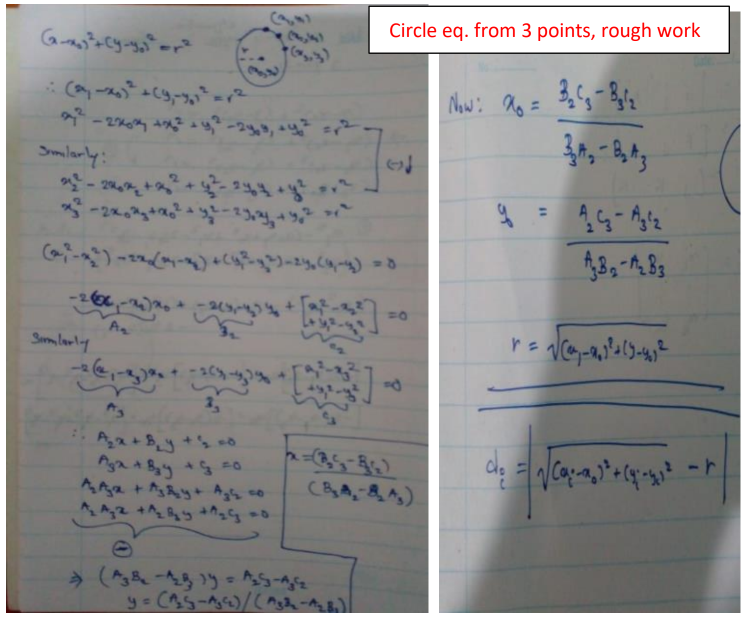


Question 01

RANSAC algo for circle detection

```python
#RANSAC circle finder
thres = 2
randomSamplingCount = 10000 #100000
# three sets of random points are selected from the given point set
#each time , one from each 3 groups will be selected (in the order) as the 3 poi
pointSet1_indexes = np.random.randint(0,len(X),(randomSamplingCount))
pointSet2_indexes = np.random.randint(0,len(X),(randomSamplingCount))
pointSet3_indexes = np.random.randint(0,len(X),(randomSamplingCount))
#points can be non-distinct -- fix it later -----------------------------------

x1 = np.zeros(randomSamplingCount)
y1 =np.zeros(randomSamplingCount)
x2 =np.zeros(randomSamplingCount)
y2 =np.zeros(randomSamplingCount)
x3 =np.zeros(randomSamplingCount)
y3 =np.zeros(randomSamplingCount)
for i in range(randomSamplingCount):
    x1[i] = X[pointSet1_indexes[i]][0]
    y1[i] = X[pointSet1_indexes[i]][1]
    x2[i] = X[pointSet2_indexes[i]][0]
    y2[i] = X[pointSet2_indexes[i]][1]
    x3[i] = X[pointSet3_indexes[i]][0]
    y3[i] = X[pointSet3_indexes[i]][1]

print("debug x1 shape",x1.shape)

#These variables are needed to calculate the cerntre coords and the radii of cir
A2 = -2*(x1-x2)
B2 = -2*(y1-y2)
C2 = x1**2-x2**2+y1**2-y2**2
A3 = -2*(x1-x3)
B3 = -2*(y1-y3)
```

Circle eq. from 3 points, rough work

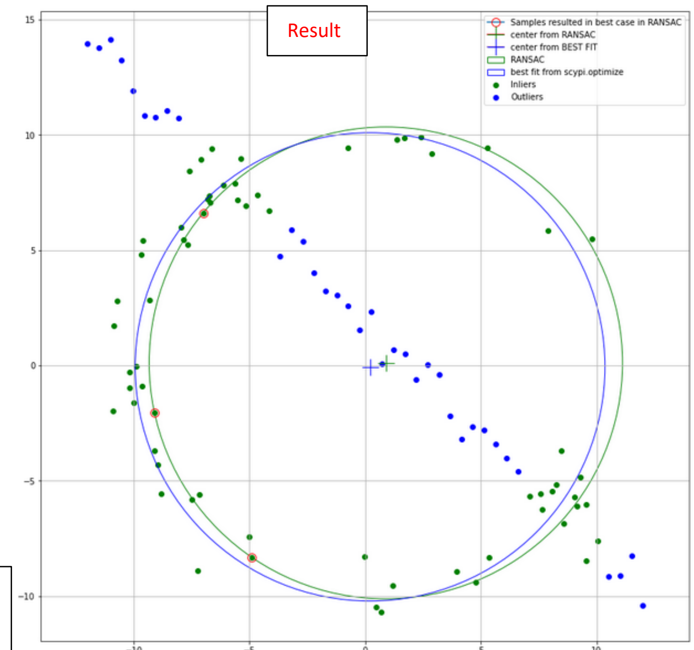


```python
x0Set = (B2*C3-B3*C2)/(B3*A2-B2*A3)
y0Set = (A2*C3-A3*C2)/(A3*B2-A2*B3)
rSet = np.sqrt((x1-x0Set)**2 + (y1-y0Set)**2)
countSet = np.zeros(len(x0Set))

print("debug xoSet shape",x0Set.shape)
#counting the inliers
for i in range(len(x0Set)):
    x0 = x0Set[i]
    y0 = y0Set[i]
    r = rSet[i]
    x = X[:,0]
    y = X[:,1]
    dSet = abs(np.sqrt((x-x0)**2+(y-y0)**2)-r) #d
    close  = dSet < thres
    count = np.sum(close)
    countSet[i] = count

mostMatchingIndex = np.argmax(countSet)
x0Best = x0Set[mostMatchingIndex]
y0Best = y0Set[mostMatchingIndex]
rBest = rSet[mostMatchingIndex]
```

Calculating inlier count for each random sample of size 3, then finding the maximum case as the solution

```
debug countSet shape (10000,)
142
equation of the circle detected
(x-x0)**2 + (y-y0)**2 = r**2
(x - (0.8931490404085495))**2 + (y - (0.1119989989460876))**2 = 10.229327367079403**2
len(X_inliers_y) 65
bestFitCircle [0.2120519825223083, -0.058760650800690546, 10.152210574797733]
```

Result




```python
def bestFitCircleFinder(x,y):

    x_m = np.mean(x)
    y_m = np.mean(y)

    def calc_R(xc, yc):
        #calculate the distance of each 2D points from the center (xc, yc)
        return np.sqrt((x-xc)**2 + (y-yc)**2)

    def f_2(c):
        # calculate the algebraic distance between the data points and the
        Ri = calc_R(*c)
        return Ri - np.mean(Ri)

    center_estimate = x_m, y_m
    center_2, ier = optimize.leastsq(f_2, center_estimate)

    xc, yc = center_2
    Ri         = calc_R(*center_2)
    R        = np.mean(Ri)

    return ([xc,yc,R])

bestFitCircle = bestFitCircleFinder(X_inliers_x,X_inliers_y)
print("bestFitCircle",bestFitCircle)
```

Best circle finder with scipy.optimize

Used Best circle finding code is from

https://scipy-cookbook.readthedocs.io/items/Least_Squares_Circle.html#Using-scipy.optimize.leastsq

Important points:

1. Instead of looping through a random index set, here I am using three random numpy vector s of indexes to select 3 points randomly for RANSAC. All the calculations are done as vectors to make the process faster.
2. 10,000 samples are taken for the RANSAC
3. Both RANSAC and best fit (least square) are almost overlapping **for threshold of 2.** Here shown is a most deviated instance I got.
4. Mistakenly did the line finding part (not asked in the assignment) and it Is in the python notebook.(GITHUB)