
Assignment 2: Line Fitting and Alignment

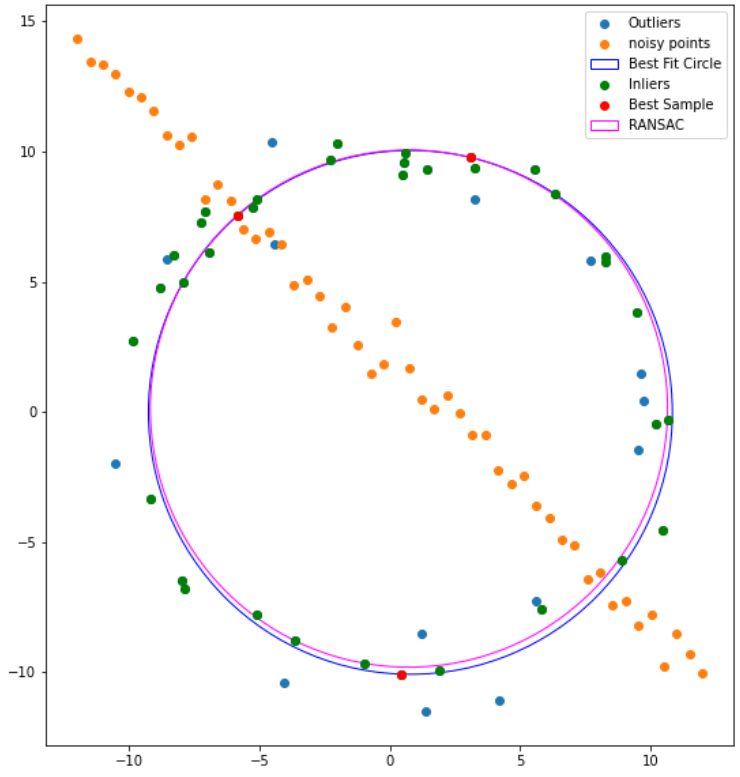
Name: Bandara D.R.K.W.M.S.D.

Index Number: 190071B

Question 1

In [12]:
```
#Question 1
#Noisy points(given code)
import numpy as np
from scipy . optimize import minimize
from scipy import linalg
import matplotlib . pyplot as plt
# np . random . seed ( 0 )
#RANSAC function
def RANSAC(X_circ,ThresholdDistance,OutlierRatio,probability,s,MaxIterations):
    bestModelX = 0
    bestModelY = 0
    bestModelR = 0
    numOfInliers = 0
    samplePoint1,samplePoint2,samplePoint3 = [0,0],[0,0],[0,0]
    inliersX = []
    inliersY = []
    cenX = []
    cenY = []
    for i in range(MaxIterations):
        randInt1,randInt2,randInt3 = np.random.randint(0,10),np.random.randint(0,10),np.random.randint(0,10)
        while randInt1==randInt2 or randInt2==randInt3 or randInt3==randInt1:
            randInt1,randInt2,randInt3 = np.random.randint(0,10),np.random.randint(0,10),np.random.randint(0,10)

        x1,y1 = X_circ[randInt1,0],X_circ[randInt1,1]
        x2,y2 = X_circ[randInt2,0],X_circ[randInt2,1]
        x3,y3 = X_circ[randInt3,0],X_circ[randInt3,1]
        f = ((x1**2-x2**2+y1**2-y2**2)*(x1-x3)-(x1**2-x3**2+y1**2-y3**2)*(x1-x2))/(2*((y1-y3)*(x1-x2)-(y1-y2)*(x1-x3)))
        g = ((x1**2-x2**2+y1**2-y2**2)*(y1-y3)-(x1**2-x3**2+y1**2-y3**2)*(y1-y2))/(2*((y1-y2)*(x1-x3)-(y1-y3)*(x1-x2)))
        c = -(x1**2+y1**2+2*g*x1+2*f*y1)
        centerX = -g
        centerY = -f
        radius = (g**2+f**2-c)**(0.5)
        count = 0
        in_x = []
        in_y = []
        for j in range(len(X_circ[:,0])):
            distance_1 = abs(((X_circ[j,0]-centerX)**2+(X_circ[j,1]-centerY)**2)**(0.5)-radius)
            if distance_1<= ThresholdDistance:
                in_x.append(X_circ[j,0])
                in_y.append(X_circ[j,1])
                count+= 1

        if count > numOfInliers:
            bestModelX = centerX
            bestModelY = centerY
            bestModelR = radius
            numOfInliers = count
            samplePoint1[0] = x1
            samplePoint1[1] = y1
            samplePoint2[0] = x2
            samplePoint2[1] = y2
            samplePoint3[0] = x3
            samplePoint3[1] = y3
            inliersX = in_x
            inliersY = in_y
        if count/len(X_circ[:,0]) >= 1-OutlierRatio:
            cenX.append(centerX)
            cenY.append(centerY)
    return bestModelX,bestModelY,bestModelR,inliersX,inliersY,cenX,cenY,samplePoint1,samplePoint2,samplePoint3
N = 100
half_n = N // 2
r = 10
s = r /16
t = np . random . uniform (0 , 2*np . pi , half_n )
n = s*np . random . randn ( half_n )
x , y = ( r + n)*np . cos ( t ) , ( r + n)*np . sin ( t )
X_circ = np . hstack ( ( x . reshape ( half_n , 1 ) , y . reshape ( half_n , 1 ) ) )
m, b = -1, 2
x = np . linspace (-12, 12 , half_n )
y = m*x + b + s*np . random . randn ( half_n )
X_line = np . hstack ( ( x . reshape ( half_n , 1 ) , y . reshape ( half_n , 1 ) ) )
X = np . vstack ( ( X_circ , X_line ) )
figure, axes = plt.subplots(figsize = (10,10))
axes.plot(X_circ[:,0],X_circ[:,1],'o',label = 'Outliers')
axes.plot(X_line[:,0],X_line[:,1],'o',label = 'noisy points')
#////////////////////////
#Best Fit circle
```

```
#//////////////////////
ThresholdDistance = 1
OutlierRatio = 0.4
probability = 0.99
s = 3
MaxIterations = int(np.log(1-probability)/np.log(1-(1-OutlierRatio)**s))

bestModelX,bestModelY,bestModelR,inliersX,inliersY,cenX,cenY,samplePoint1,samplePoint2,samplePoint3 = RANSAC(X_circ,ThresholdDista

circle = plt.Circle((bestModelX,bestModelY),bestModelR,fill = False,color = 'blue',label = 'Best Fit Circle')
axes.set_aspect( 1 )
axes.add_artist( circle )
axes.plot(inliersX,inliersY,'o',color='green',label = 'Inliers')
axes.plot(np.array([samplePoint1[0],samplePoint2[0],samplePoint3[0]]), np.array([samplePoint1[1],samplePoint2[1],samplePoint3[1]])

#//////////////////////////////////
#RANSAC Circle
#//////////////////////////////////

inliersX = np.array(inliersX).reshape(len(inliersX),1)
inliersY = np.array(inliersY).reshape(len(inliersY),1)
X_Circ_ = np.concatenate((inliersX, inliersY), axis=1)
bestModelX_,bestModelY_,bestModelR_,inliersX_,inliersY_,cenX_,cenY_,samplePoint1_,samplePoint2_,samplePoint3_ = RANSAC(X_Circ_,Thr
circle_ = plt.Circle((bestModelX_,bestModelY_),bestModelR_,fill = False,color = 'magenta',label = 'RANSAC')
axes.set_aspect( 1 )
axes.add_artist(circle_)
plt.legend()
plt.show()
```



Question 2

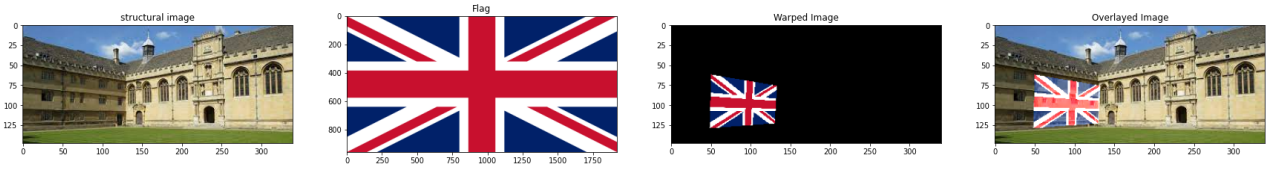
In [6]:

```
#Question 2
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
#Mouse clicking
def mousePoint(event, x,y,flags,params):
    if event == cv.EVENT_LBUTTONDOWN:
        print(x,y)
        imgPoints.append([x,y,1])

img = cv.imread('clg.jpg',cv.IMREAD_ANYCOLOR)
cv.imshow("IMG",img)
imgPoints = []
cv.setMouseCallback("IMG",mousePoint)

cv.waitKey(0)
imgFlag = cv.imread('flag.png',cv.IMREAD_ANYCOLOR)
flagheight,flagWidth = imgFlag.shape[0],imgFlag.shape[1]
flagPoints = np.array([[0,0,1],[flagWidth,0,1],[flagWidth,flagheight,1],[0,flagheight,1]])
imgPoints = np.array(imgPoints)
A = np.array([[0,0,0,flagPoints[0][0],flagPoints[0][1],flagPoints[0][2],-imgPoints[0][1]*flagPoints[0][0],-imgPoints[0][1]*flagPoi
[flagPoints[0][0],flagPoints[0][1],flagPoints[0][2],0,0,0,-imgPoints[0][0]*flagPoints[0][0],-imgPoints[0][0]*flagPoints[0][1],-img
[0,0,0,flagPoints[1][0],flagPoints[1][1],flagPoints[1][2],-imgPoints[1][1]*flagPoints[1][0],-imgPoints[1][1]*flagPoints[1][1],-img
[flagPoints[1][0],flagPoints[1][1],flagPoints[1][2],0,0,0,-imgPoints[1][0]*flagPoints[1][0],-imgPoints[1][0]*flagPoints[1][1],-img
```

```
[0,0,0,flagPoints[2][0],flagPoints[2][1],flagPoints[2][2],-imgPoints[2][1]*flagPoints[2][0],-imgPoints[2][1]*flagPoints[2][1],-img
[flagPoints[2][0],flagPoints[2][1],flagPoints[2][2],0,0,0,-imgPoints[2][0]*flagPoints[2][0],-imgPoints[2][0]*flagPoints[2][1],-img
[0,0,0,flagPoints[3][0],flagPoints[3][1],flagPoints[3][2],-imgPoints[3][1]*flagPoints[3][0],-imgPoints[3][1]*flagPoints[3][1],-img
[flagPoints[3][0],flagPoints[3][1],flagPoints[3][2],0,0,0,-imgPoints[3][0]*flagPoints[3][0],-imgPoints[3][0]*flagPoints[3][1],-img
],dtype=np.float64)

AAT = A.T @ A
w, v = np.linalg.eig(AAT)
minIndex = np.where(w == np.min(w))[0][0]
H = v[:, np.argmin(w)]
H = H.reshape((3,3))
print('H= ',H)

rows,cols,ch = img.shape
im1to4 = cv.warpPerspective(imgFlag,H,(cols,rows))#(imgPoints[2][0]-imgPoints[0][0],imgPoints[2][1]-imgPoints[0][1]))

overlayImg = cv.add(img, im1to4)
# result = cv.addWeighted(imTemp, 0.3, im1to4, 0.7, 0.0)
fig,ax = plt.subplots(1,4,figsize = (30,10))
im1to4 = cv.cvtColor(im1to4, cv.COLOR_BGR2RGB)
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
imgFlag = cv.cvtColor(imgFlag, cv.COLOR_BGR2RGB)
overlayImg = cv.cvtColor(overlayImg, cv.COLOR_BGR2RGB)
ax[0].imshow(img)
ax[0].set_title("structural image")
ax[1].imshow(imgFlag)
ax[1].set_title("Flag")
ax[2].imshow(im1to4)
ax[2].set_title("Warped Image")
ax[3].imshow(overlayImg)
ax[3].set_title("Overlayed Image")
```

```
50 62
133 76
130 123
48 129
H=  [[-9.15852770e-04  8.64536808e-06 -6.27702425e-01]
 [-3.04769493e-04 -9.23223142e-04 -7.78350999e-01]
 [-2.80565204e-06 -3.64768812e-07 -1.25540484e-02]]
```

Out[6]: Text(0.5, 1.0, 'Overlayed Image')



Question 3

In [7]:
```
#Question 3
#SIFT features
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv

img1 = cv.imread('img1.ppm')
img2 = cv.imread('img4.ppm')

img1 = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
img2 = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)

sift = cv.xfeatures2d.SIFT_create()
keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(img2,None)

FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)
flann = cv.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(descriptors_1,descriptors_2,k=2)
pts1 = []
pts2 = []

for i,(m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        pts2.append(keypoints_2[m.trainIdx].pt)
        pts1.append(keypoints_1[m.queryIdx].pt)

pts1 = np.int32(pts1)
pts2 = np.int32(pts2)
```

In [8]:
```
src = np.concatenate((pts1,np.ones((pts1.shape[0],1))),axis=1)
dst = np.concatenate((pts2,np.ones((pts2.shape[0],1))),axis=1)
numOfIn = 0
inS = []
inD = []
for J in range(100):
    #generate 4 random numbers for choose 4 points
```

```
    randInt1,randInt2,randInt3,randInt4 = np.random.randint(0,2000)%(src.shape[0]),np.random.randint(0,2000)%(src.shape[0]),np.ran
    while randInt1==randInt2 or randInt2==randInt3 or randInt3==randInt1 or randInt4==randInt1 or randInt3==randInt4 or randInt2==
        randInt1,randInt2,randInt3,randInt4 = np.random.randint(0,2000)%(src.shape[0]),np.random.randint(0,2000)%(src.shape[0]),np


    src1 = np.array([src[randInt1],src[randInt2],src[randInt3],src[randInt4]])
    dst1 = np.array([dst[randInt1],dst[randInt2],dst[randInt3],dst[randInt4]])
    A = []
    for i in range(src1.shape[0]):
        A.append([src1[i][0],src1[i][1],src1[i][2],0,0,0,-dst1[i][0]*src1[i][0],-dst1[i][0]*src1[i][1],-dst1[i][0]*src1[i][2]])
        A.append([0,0,0,src1[i][0],src1[i][1],src1[i][2],-dst1[i][1]*src1[i][0],-dst1[i][1]*src1[i][1],-dst1[i][1]*src1[i][2]])

    #find H
    A = np.array(A)
    U, S, Vh = np.linalg.svd(A)
    L = Vh[-1, :] / Vh[-1, -1]
    H = L.reshape(3, 3)
    #find number of inliers
    epsilon = 20
    count = 0
    inlierss = []
    inliersd = []
    for k in range(src.shape[0]):
        x_ = H.dot((src[k,:].reshape(3, 1)))
        x_ = x_ [:2].reshape(1, 2)[0]
        x__ = (dst[k,0:2].reshape(1, 2)[0]-x_)
        if  abs(x__[0])< epsilon and abs(x__[1]) < epsilon:

            count += 1
            inlierss.append(src[k,:])
            inliersd.append(dst[k,:])

    #select occassion of  highest number of inliers
    if numOfIn < count:
        numOfIn = count
        inS = inlierss.copy()
        inD = inliersd.copy()

inS = np.array(inS.copy())
inD = np.array(inD.copy())

src1 = inS.copy()
dst1 = inD.copy()
#Calculate H
A = []
for i in range(inS.shape[0]):
    A.append([src1[i][0],src1[i][1],src1[i][2],0,0,0,-dst1[i][0]*src1[i][0],-dst1[i][0]*src1[i][1],-dst1[i][0]*src1[i][2]])
    A.append([0,0,0,src1[i][0],src1[i][1],src1[i][2],-dst1[i][1]*src1[i][0],-dst1[i][1]*src1[i][1],-dst1[i][1]*src1[i][2]])

A = np.array(A)
U, S, Vh = np.linalg.svd(A)
L = Vh[-1, :] / Vh[-1, -1]
H = L.reshape(3, 3)

print('H= ',H)
```

```
H=  [[ 6.41509065e-01  6.62645011e-01 -2.83483192e+01]
 [-1.62783256e-01  9.29720278e-01  1.53982727e+02]
 [ 3.72700100e-04 -5.30961156e-05  1.00000000e+00]]
```

In [9]:

```
im1to5 = cv.warpPerspective(img1,H,(1000,1000))#(imgPoints[2][0]-imgPoints[0][0],imgPoints[2][1]-imgPoints[0][1]))
fig,ax = plt.subplots(1,3,figsize = (20,5))
im1to5 = cv.cvtColor(im1to5, cv.COLOR_BGR2RGB)
img1 = cv.cvtColor(img1, cv.COLOR_BGR2RGB)
img2 = cv.cvtColor(img2, cv.COLOR_BGR2RGB)

ax[0].imshow(img1)
ax[0].set_title("Image 1")
ax[1].imshow(img2)
ax[1].set_title("Image 2")
ax[2].imshow(im1to5)
ax[2].set_title("Warped Image")
```

Out[9]:

```
Text(0.5, 1.0, 'Warped Image')
```