Department of Electronics and Telecommunication Engineering
University of Moratuwa

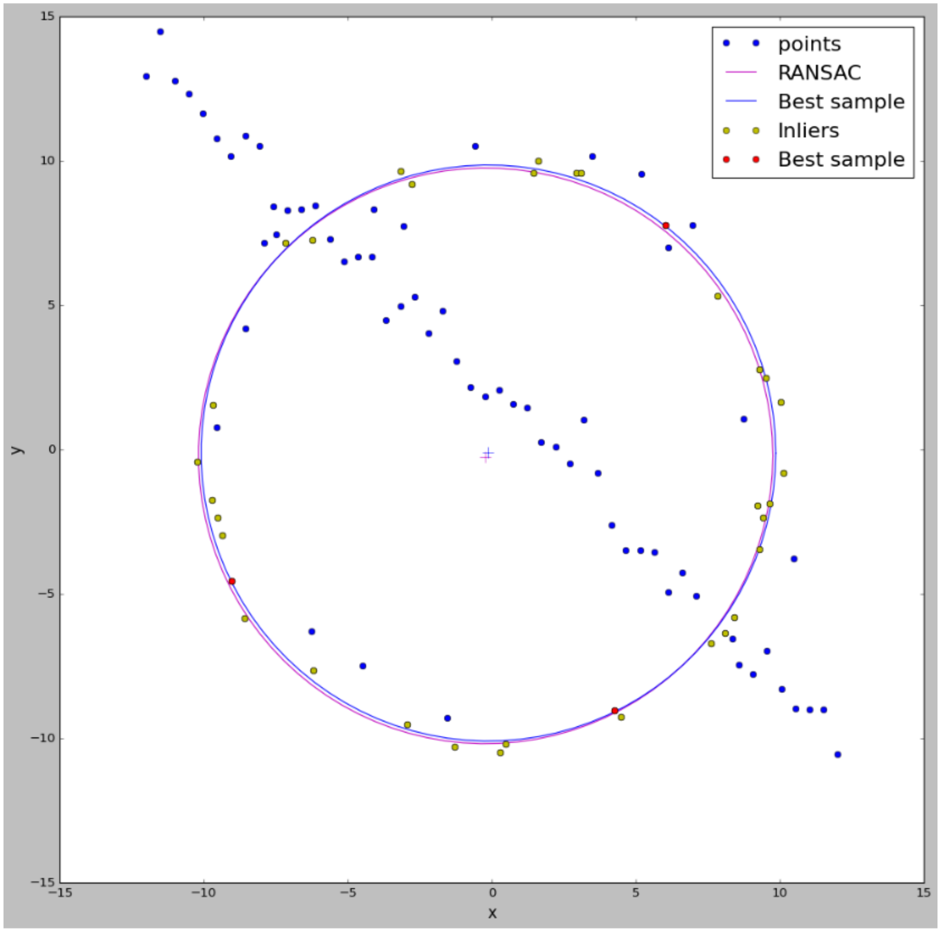
EN2550 – Fundamentals of Image Processing and Machine Vision
Assignment 2

Pathirana R.P.U.A. – 190432J

https://github.com/Uvin99/EN2550_Assignment2

Question 01

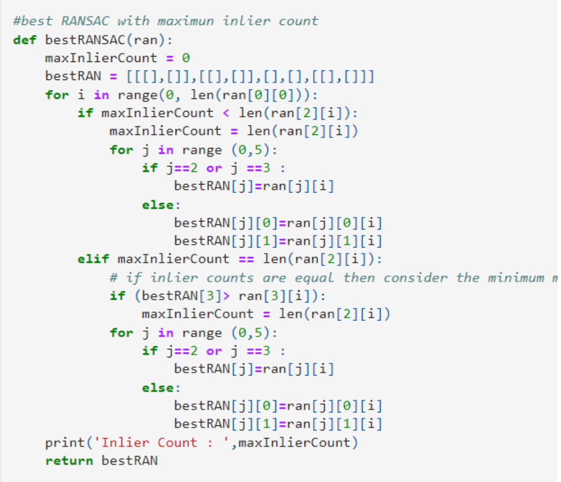
Plot



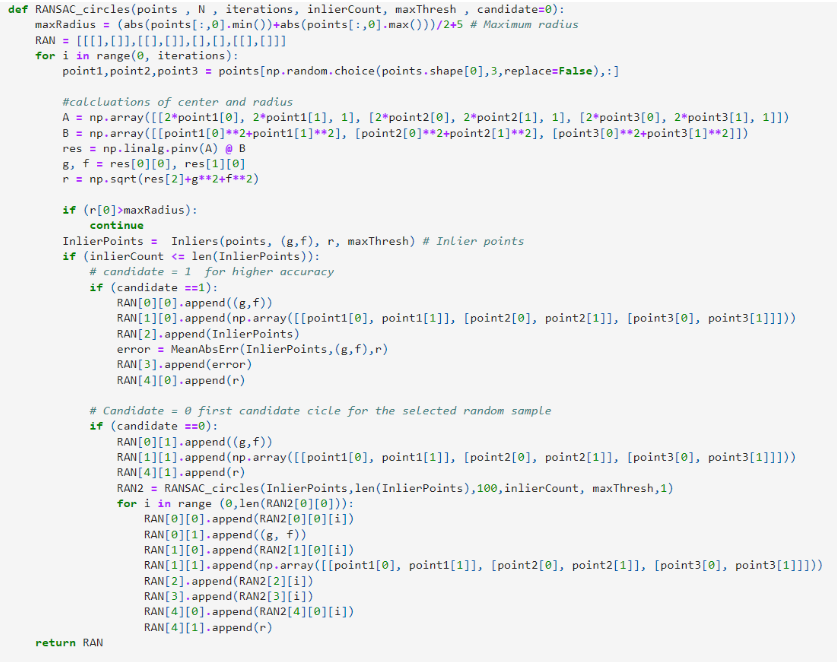
From the above plot, we can observe that the circle estimated from the best sample and the circle estimated using the RANSAC algorithm are almost the same. The distance between the two centers is quite small.

Code

bestRANSAC function is used to get the best RANSAC with maximum inlier count. RANSAC_circles function is used to get the RANSAC circles. Some functions are defined to get co-ordinate points of circle, mean absolute error, and to detect inlier points. Important parts of the code are displayed below and the full code can be found in github repository.



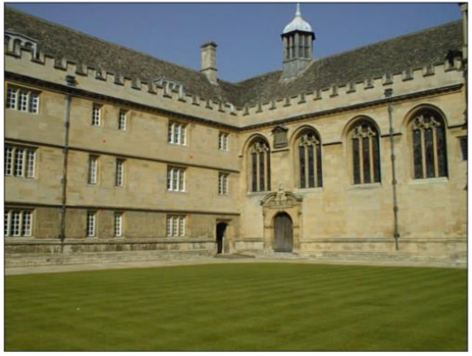
```python
#best RANSAC with maximun inlier count
def bestRANSAC(ran):
    maxInlierCount = 0
    bestRAN = [[[],[]],[[],[]],[],[],[[],[]]]
    for i in range(0, len(ran[0][0])):
        if maxInlierCount < len(ran[2][i]):
            maxInlierCount = len(ran[2][i])
            for j in range (0,5):
                if j==2 or j ==3 :
                    bestRAN[j]=ran[j][i]
                else:
                    bestRAN[j][0]=ran[j][0][i]
                    bestRAN[j][1]=ran[j][1][i]
        elif maxInlierCount == len(ran[2][i]):
            # if inlier counts are equal then consider the minimum m
            if (bestRAN[3]> ran[3][i]):
                maxInlierCount = len(ran[2][i])
            for j in range (0,5):
                if j==2 or j ==3 :
                    bestRAN[j]=ran[j][i]
                else:
                    bestRAN[j][0]=ran[j][0][i]
                    bestRAN[j][1]=ran[j][1][i]
    print('Inlier Count : ',maxInlierCount)
    return bestRAN
```



```python
def RANSAC_circles(points , N , iterations, inlierCount, maxThresh , candidate=0):
    maxRadius = (abs(points[:,0].min())+abs(points[:,0].max()))/2+5 # Maximum radius
    RAN = [[[],[]],[[],[]],[],[],[[],[]]]
    for i in range(0, iterations):
        point1,point2,point3 = points[np.random.choice(points.shape[0],3,replace=False),:]

        #calcluations of center and radius
        A = np.array([[2*point1[0], 2*point1[1], 1], [2*point2[0], 2*point2[1], 1], [2*point3[0], 2*point3[1], 1]])
        B = np.array([[point1[0]**2+point1[1]**2], [point2[0]**2+point2[1]**2], [point3[0]**2+point3[1]**2]])
        res = np.linalg.pinv(A) @ B
        g, f = res[0][0], res[1][0]
        r = np.sqrt(res[2]+g**2+f**2)

        if (r[0]>maxRadius):
            continue
        InlierPoints =  Inliers(points, (g,f), r, maxThresh) # Inlier points
        if (inlierCount <= len(InlierPoints)):
            # candidate = 1  for higher accuracy
            if (candidate ==1):
                RAN[0][0].append((g,f))
                RAN[1][0].append(np.array([[point1[0], point1[1]], [point2[0], point2[1]], [point3[0], point3[1]]]))
                RAN[2].append(InlierPoints)
                error = MeanAbsErr(InlierPoints,(g,f),r)
                RAN[3].append(error)
                RAN[4][0].append(r)

            # Candidate = 0 first candidate cicle for the selected random sample
            if (candidate ==0):
                RAN[0][1].append((g,f))
                RAN[1][1].append(np.array([[point1[0], point1[1]], [point2[0], point2[1]], [point3[0], point3[1]]]))
                RAN[4][1].append(r)
                RAN2 = RANSAC_circles(InlierPoints,len(InlierPoints),100,inlierCount, maxThresh,1)
                for i in range (0,len(RAN2[0][0])):
                    RAN[0][0].append(RAN2[0][0][i])
                    RAN[0][1].append((g, f))
                    RAN[1][0].append(RAN2[1][0][i])
                    RAN[1][1].append(np.array([[point1[0], point1[1]], [point2[0], point2[1]], [point3[0], point3[1]]]))
                    RAN[2].append(RAN2[2][i])
                    RAN[3].append(RAN2[3][i])
                    RAN[4][0].append(RAN2[4][0][i])
                    RAN[4][1].append(r)
    return RAN
```

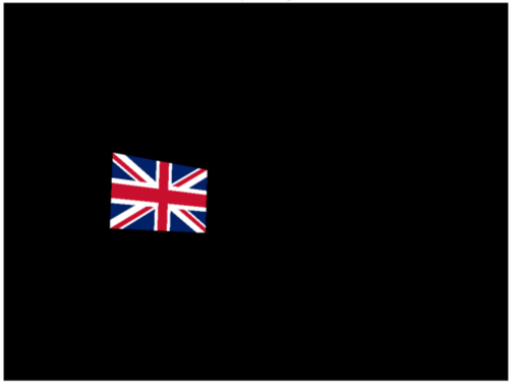
Question2 - Image Results

Architectural Image

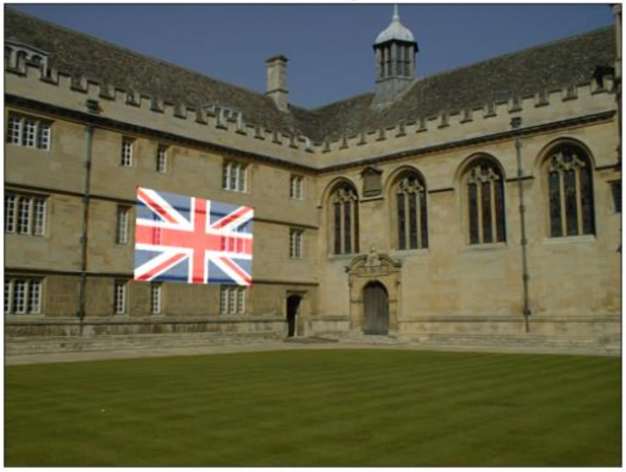


Flag Image

Warped Flag

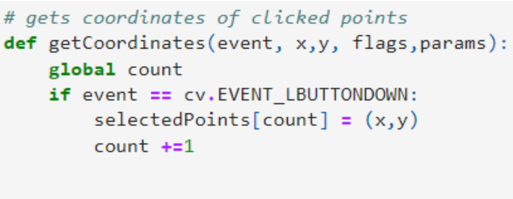


Blended Image

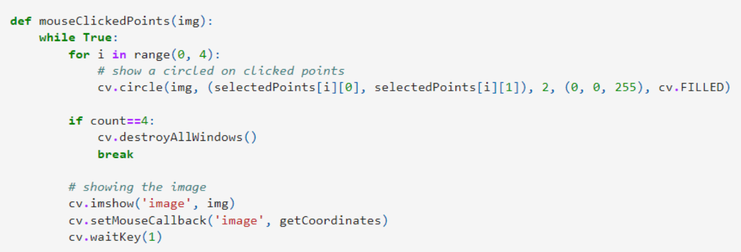


Code

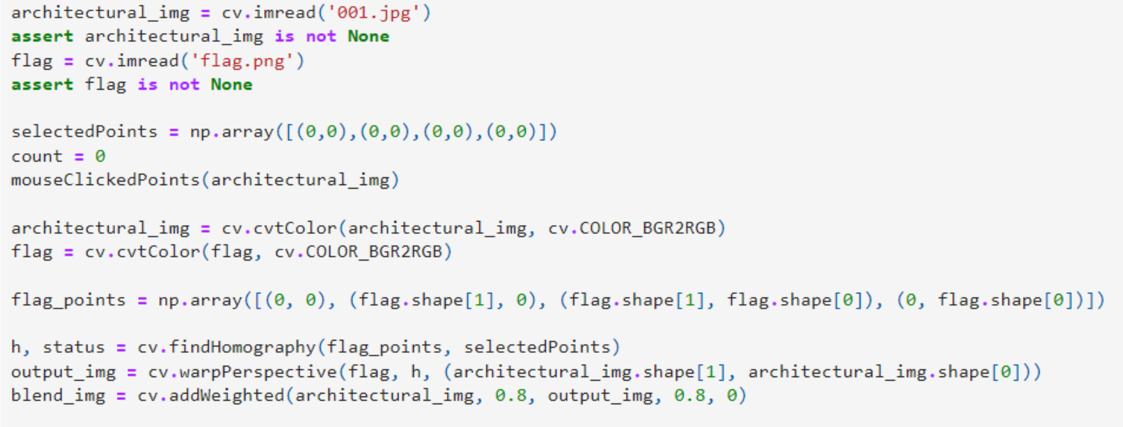
getCoordinates function is defined to get the coordinates of mouse clicked points. The global variable “count” is updated to detect whether 4 points are clicked. mouseClickedPoints function is used to show red dots on clicked points and close the image after 4 points are clicked. Important parts of the code are shown below.



```
# gets coordinates of clicked points
def getCoordinates(event, x,y, flags,params):
    global count
    if event == cv.EVENT_LBUTTONDOWN:
        selectedPoints[count] = (x,y)
        count +=1
```

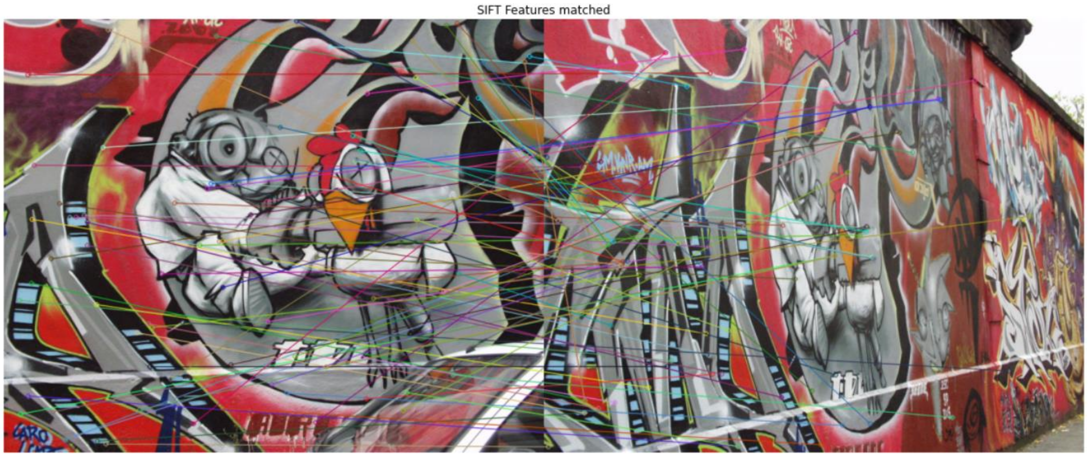


```
def mouseClickedPoints(img):
    while True:
        for i in range(0, 4):
            # show a circled on clicked points
            cv.circle(img, (selectedPoints[i][0], selectedPoints[i][1]), 2, (0, 0, 255), cv.FILLED)

        if count==4:
            cv.destroyAllWindows()
            break

        # showing the image
        cv.imshow('image', img)
        cv.setMouseCallback('image', getCoordinates)
        cv.waitKey(1)
```



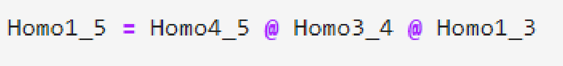
```
architectural_img = cv.imread('001.jpg')
assert architectural_img is not None
flag = cv.imread('flag.png')
assert flag is not None

selectedPoints = np.array([(0,0),(0,0),(0,0),(0,0)])
count = 0
mouseClickedPoints(architectural_img)

architectural_img = cv.cvtColor(architectural_img, cv.COLOR_BGR2RGB)
flag = cv.cvtColor(flag, cv.COLOR_BGR2RGB)

flag_points = np.array([(0, 0), (flag.shape[1], 0), (flag.shape[1], flag.shape[0]), (0, flag.shape[0])])

h, status = cv.findHomography(flag_points, selectedPoints)
output_img = cv.warpPerspective(flag, h, (architectural_img.shape[1], architectural_img.shape[0]))
blend_img = cv.addWeighted(architectural_img, 0.8, output_img, 0.8, 0)
```

Question 3



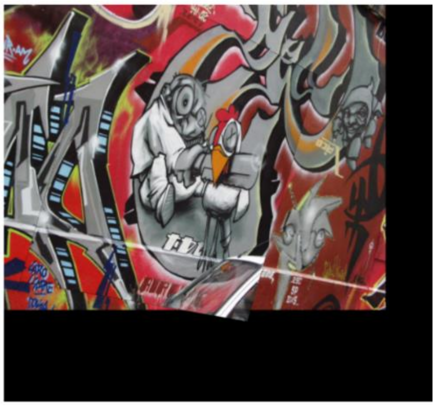
SIFT Features matched

From the above result, we can observe that there is a very less number of matches when directly matching img1.ppm to img5.ppm. Therefore stitching img1 directly onto img5 did not give the expected result. Homography transforms of img1 to img3, img3 to img4, and img4 to img5 were obtained. Then the homography transform of img1 to img5 was calculated as follows.

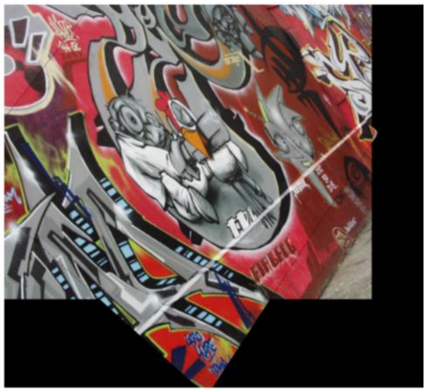


```
Homo1_5 = Homo4_5 @ Homo3_4 @ Homo1_3
```

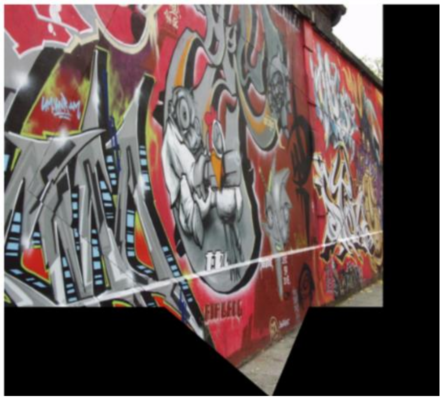
matchSIFT function was used to find the good matches and then RANSAC was used to filter out the right matches.



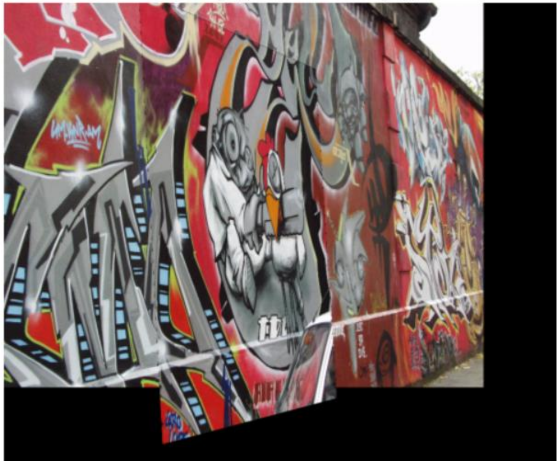
img1.ppm over img3.ppm



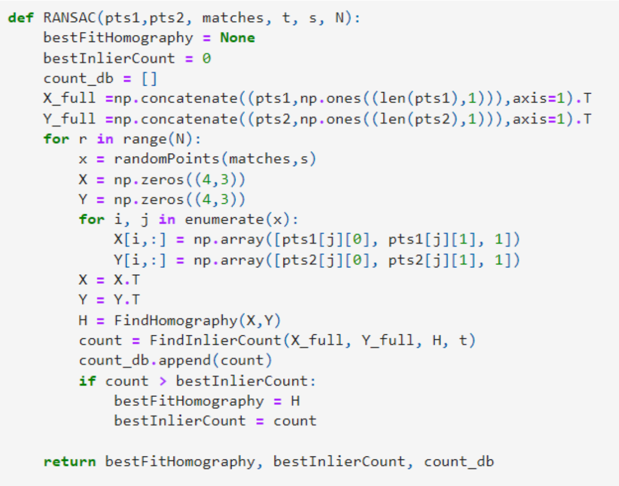
img3.ppm over img4.ppm



img4.ppm over img5.ppm

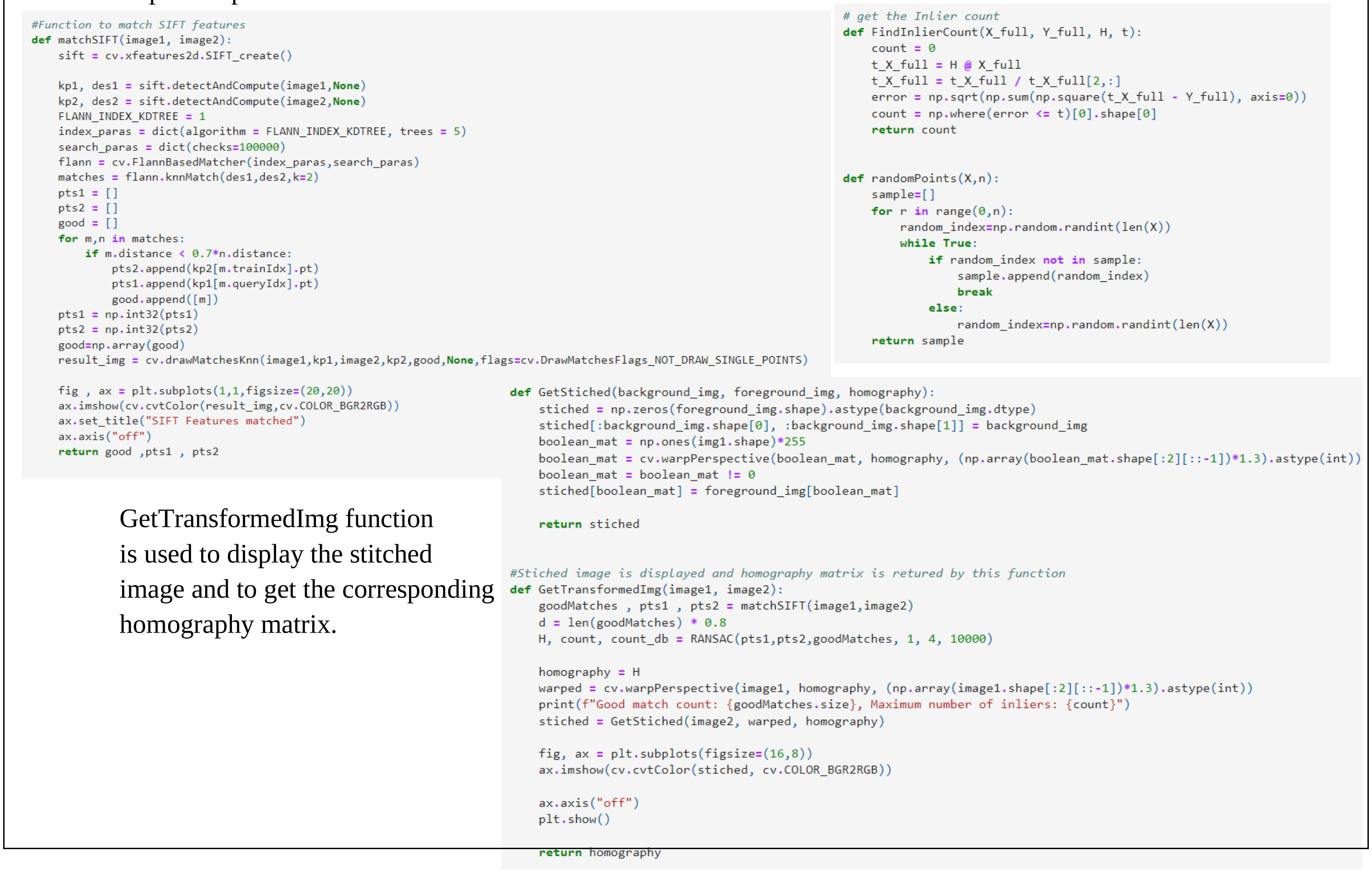


Img1.ppm stiched onto Img5.ppm



```
def RANSAC(pts1,pts2, matches, t, s, N):
    bestFitHomography = None
    bestInlierCount = 0
    count_db = []
    X_full =np.concatenate((pts1,np.ones((len(pts1),1))),axis=1).T
    Y_full =np.concatenate((pts2,np.ones((len(pts2),1))),axis=1).T
    for r in range(N):
        x = randomPoints(matches,s)
        X = np.zeros((4,3))
        Y = np.zeros((4,3))
        for i, j in enumerate(x):
            X[i,:] = np.array([pts1[j][0], pts1[j][1], 1])
            Y[i,:] = np.array([pts2[j][0], pts2[j][1], 1])
        X = X.T
        Y = Y.T
        H = FindHomography(X,Y)
        count = FindInlierCount(X_full, Y_full, H, t)
        count_db.append(count)
        if count > bestInlierCount:
            bestFitHomography = H
            bestInlierCount = count

    return bestFitHomography, bestInlierCount, count_db
```

Important parts of the code



```
#Function to match SIFT features
def matchSIFT(image1, image2):
    sift = cv.xfeatures2d.SIFT_create()

    kp1, des1 = sift.detectAndCompute(image1,None)
    kp2, des2 = sift.detectAndCompute(image2,None)
    FLANN_INDEX_KDTREE = 1
    index_paras = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
    search_paras = dict(checks=100000)
    flann = cv.FlannBasedMatcher(index_paras,search_paras)
    matches = flann.knnMatch(des1,des2,k=2)
    pts1 = []
    pts2 = []
    good = []
    for m,n in matches:
        if m.distance < 0.7*n.distance:
            pts2.append(kp2[m.trainIdx].pt)
            pts1.append(kp1[m.queryIdx].pt)
            good.append([m])
    pts1 = np.int32(pts1)
    pts2 = np.int32(pts2)
    good=np.array(good)
    result_img = cv.drawMatchesKnn(image1,kp1,image2,kp2,good,None,flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

    fig , ax = plt.subplots(1,1,figsize=(20,20))
    ax.imshow(cv.cvtColor(result_img,cv.COLOR_BGR2RGB))
    ax.set_title("SIFT Features matched")
    ax.axis("off")
    return good ,pts1 , pts2
```

```
# get the Inlier count
def FindInlierCount(X_full, Y_full, H, t):
    count = 0
    t_X_full = H @ X_full
    t_X_full = t_X_full / t_X_full[2,:]
    error = np.sqrt(np.sum(np.square(t_X_full - Y_full), axis=0))
    count = np.where(error <= t)[0].shape[0]
    return count


def randomPoints(X,n):
    sample=[]
    for r in range(0,n):
        random_index=np.random.randint(len(X))
        while True:
            if random_index not in sample:
                sample.append(random_index)
                break
            else:
                random_index=np.random.randint(len(X))
    return sample
```

```
def GetStiched(background_img, foreground_img, homography):
    stiched = np.zeros(foreground_img.shape).astype(background_img.dtype)
    stiched[:background_img.shape[0], :background_img.shape[1]] = background_img
    boolean_mat = np.ones(img1.shape)*255
    boolean_mat = cv.warpPerspective(boolean_mat, homography, (np.array(boolean_mat.shape[:2][::-1])*1.3).astype(int))
    boolean_mat = boolean_mat != 0
    stiched[boolean_mat] = foreground_img[boolean_mat]

    return stiched


#Stiched image is displayed and homography matrix is retured by this function
def GetTransformedImg(image1, image2):
    goodMatches , pts1 , pts2 = matchSIFT(image1,image2)
    d = len(goodMatches) * 0.8
    H, count, count_db = RANSAC(pts1,pts2,goodMatches, 1, 4, 10000)

    homography = H
    warped = cv.warpPerspective(image1, homography, (np.array(image1.shape[:2][::-1])*1.3).astype(int))
    print(f"Good match count: {goodMatches.size}, Maximum number of inliers: {count}")
    stiched = GetStiched(image2, warped, homography)

    fig, ax = plt.subplots(figsize=(16,8))
    ax.imshow(cv.cvtColor(stiched, cv.COLOR_BGR2RGB))

    ax.axis("off")
    plt.show()

    return homography
```

GetTransformedImg function is used to display the stitched image and to get the corresponding homography matrix.