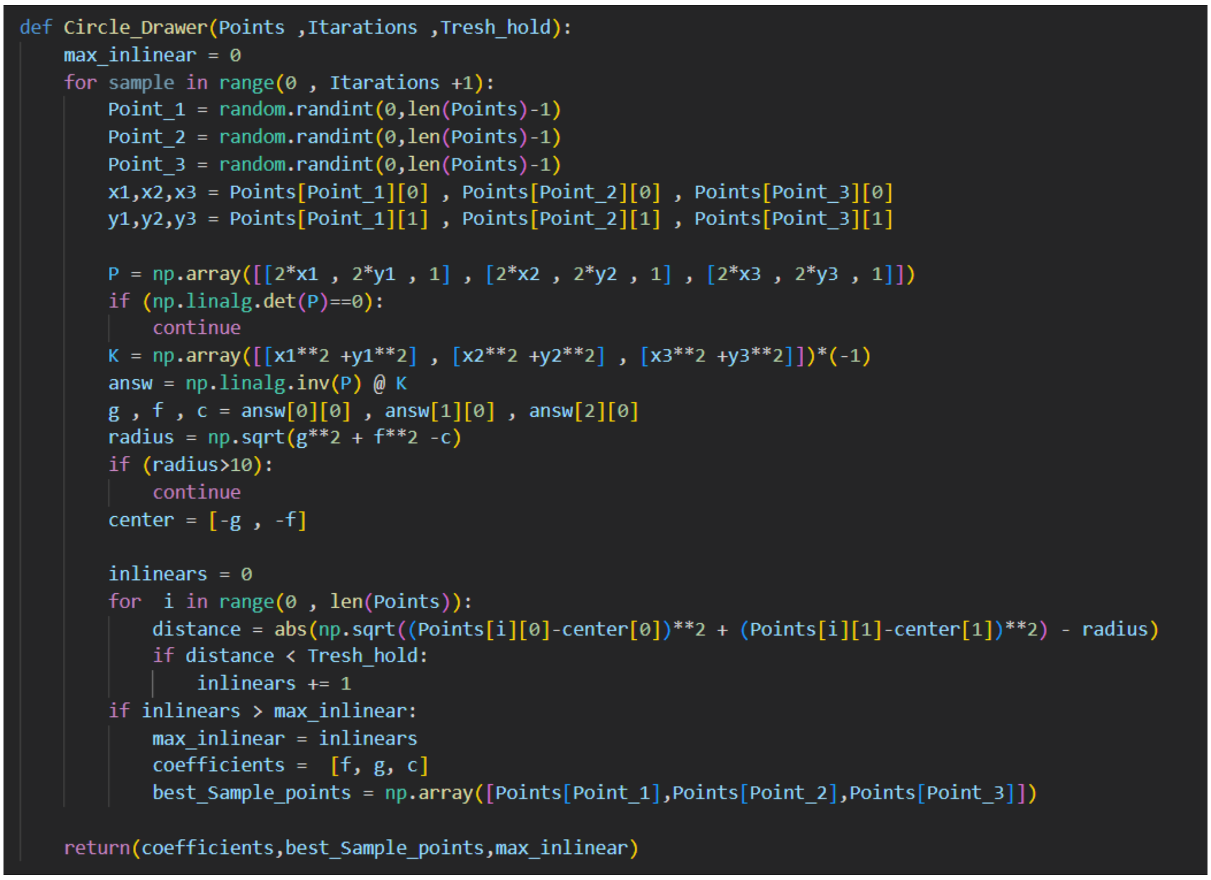
Aakkash.G – 190001M

EN 2550 Assignment 2 on Fitting and Alignment

Question 01

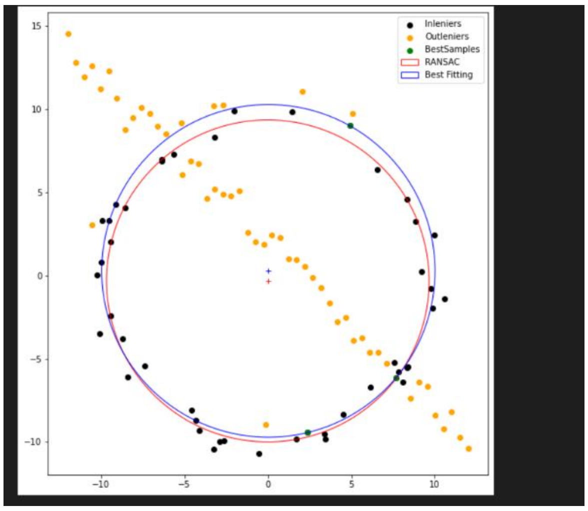
In order to draw a circle we need minimum of 3 points as per the equation $x^2 + y^2 + 2gx + 2fy + c = 0$ we need to find g ,f,c. As per the requirement of the question Ransac algoritham Circle Drawer is defined to select 3 (x,y) coordinates among the given noisy points to calculate g,f,c in every loop and return the best of the sample points & inliers. Using Circle_Drawer function number of samples need for accurate circle, circle coefficients and samples and inliers are determined.



```python
def Circle_Drawer(Points ,Itarations ,Tresh_hold):
    max_inlinear = 0
    for sample in range(0 , Itarations +1):
        Point_1 = random.randint(0,len(Points)-1)
        Point_2 = random.randint(0,len(Points)-1)
        Point_3 = random.randint(0,len(Points)-1)
        x1,x2,x3 = Points[Point_1][0] , Points[Point_2][0] , Points[Point_3][0]
        y1,y2,y3 = Points[Point_1][1] , Points[Point_2][1] , Points[Point_3][1]

        P = np.array([[2*x1 , 2*y1 , 1] , [2*x2 , 2*y2 , 1] , [2*x3 , 2*y3 , 1]])
        if (np.linalg.det(P)==0):
            continue
        K = np.array([[x1**2 +y1**2] , [x2**2 +y2**2] , [x3**2 +y3**2]])*(-1)
        answ = np.linalg.inv(P) @ K
        g , f , c = answ[0][0] , answ[1][0] , answ[2][0]
        radius = np.sqrt(g**2 + f**2 -c)
        if (radius>10):
            continue
        center = [-g , -f]

        inlinears = 0
        for  i in range(0 , len(Points)):
            distance = abs(np.sqrt((Points[i][0]-center[0])**2 + (Points[i][1]-center[1])**2) - radius)
            if distance < Tresh_hold:
                inlinears += 1
        if inlinears > max_inlinear:
            max_inlinear = inlinears
            coefficients =  [f, g, c]
            best_Sample_points = np.array([Points[Point_1],Points[Point_2],Points[Point_3]])

    return(coefficients,best_Sample_points,max_inlinear)
```

Then the best fitting circle was plotted by considering **Inlier points of the RANSAC circle.** But in this case the circle is more accurate since this time the selected samples are only from the best points that form a circle.

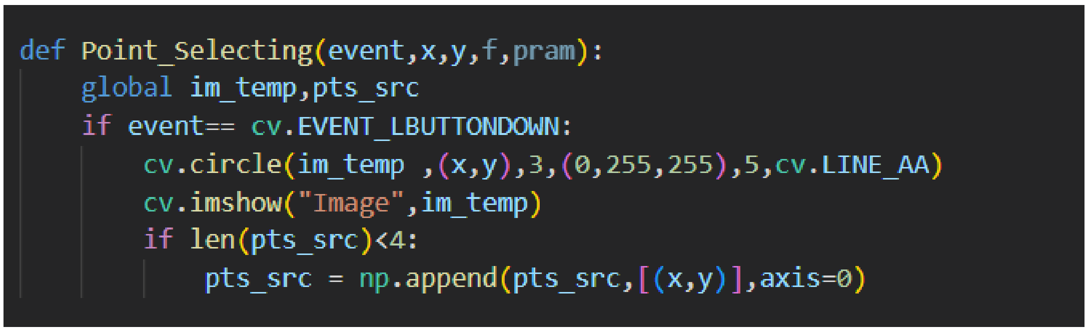
The final circle is drawn with the randomly chosen 3 points that give maximum number of inliers. In this question, the best fitting circle has 46 inlier count.



Question 02

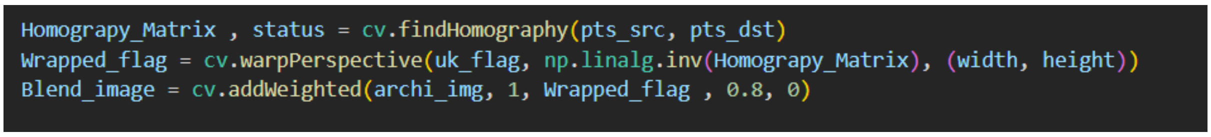
So in this question we have to superimpose a image onto another image by using a homography matrix. So as the the first step homography matrix of the the architextural image and the UK flag should be found.

Using the Point_Selecting Function the users can select four preferred points in the architectural image using a mouse click in the open in a new window. So with the yellow spots marking the selected sub plane of the architectural image.

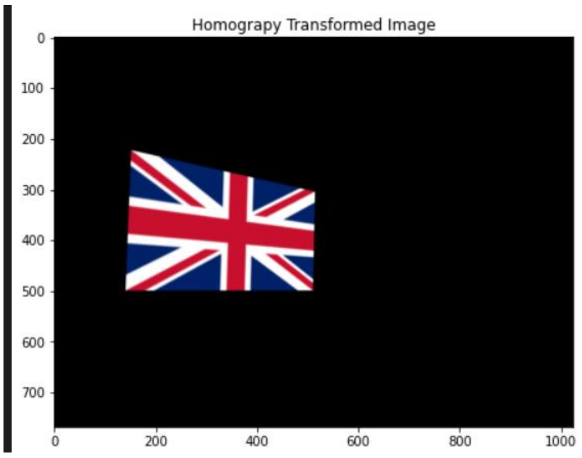


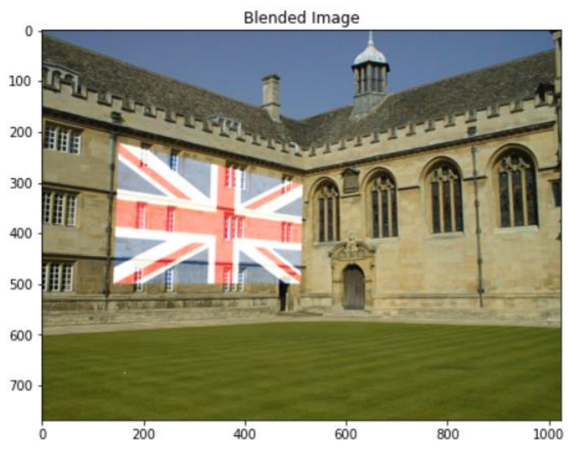
```python
def Point_Selecting(event,x,y,f,pram):
    global im_temp,pts_src
    if event== cv.EVENT_LBUTTONDOWN:
        cv.circle(im_temp ,(x,y),3,(0,255,255),5,cv.LINE_AA)
        cv.imshow("Image",im_temp)
        if len(pts_src)<4:
            pts_src = np.append(pts_src,[(x,y)],axis=0)
```

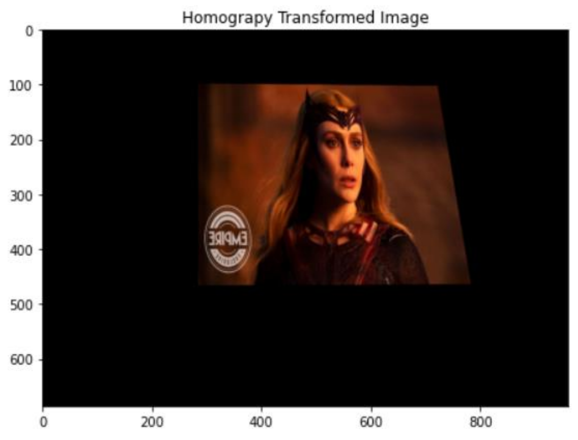
By using the inverse of the homography matrix which can be found by inbuilt open CV Function findhomography the UK flag will be wrapped using wrap perspective function and blended using add weighted function into the subplane selected by the user

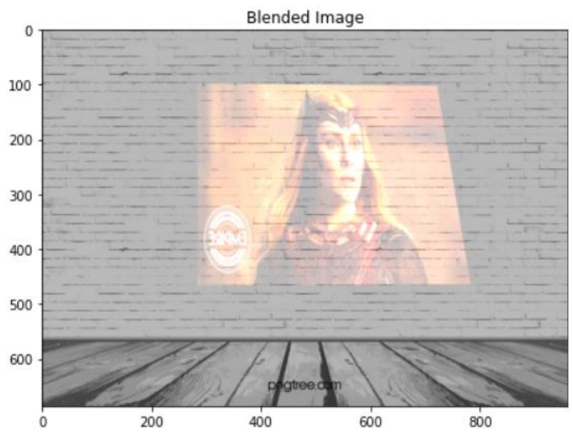


```python
Homograpy_Matrix , status = cv.findHomography(pts_src, pts_dst)
Wrapped_flag = cv.warpPerspective(uk_flag, np.linalg.inv(Homograpy_Matrix), (width, height))
Blend_image = cv.addWeighted(archi_img, 1, Wrapped_flag , 0.8, 0)
```



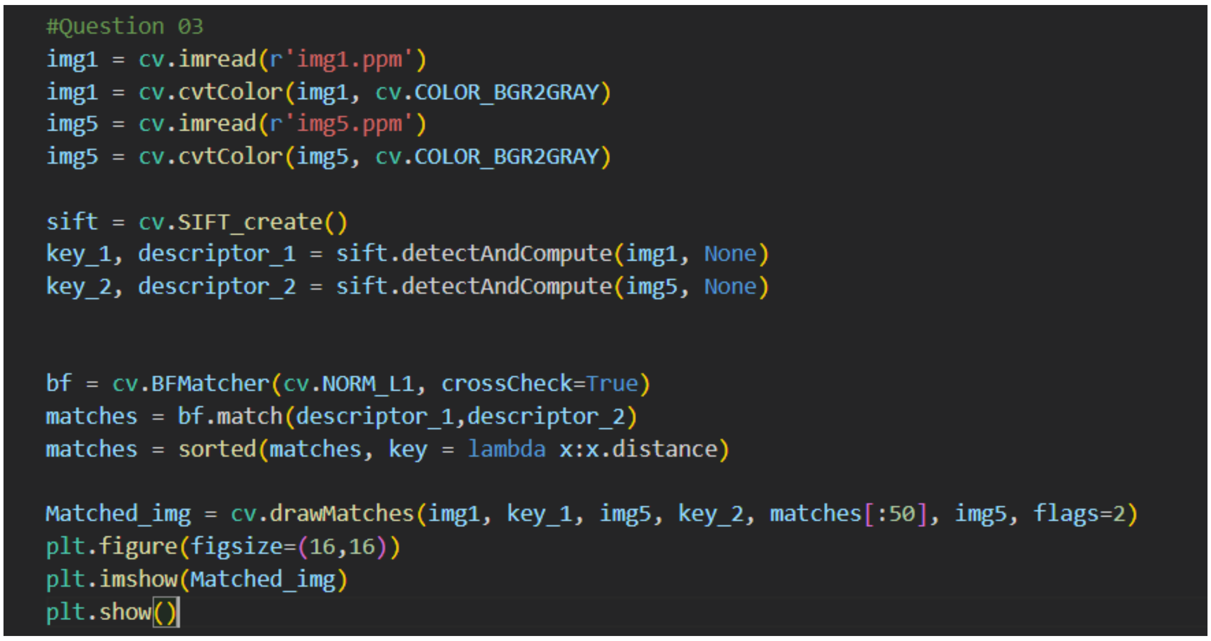






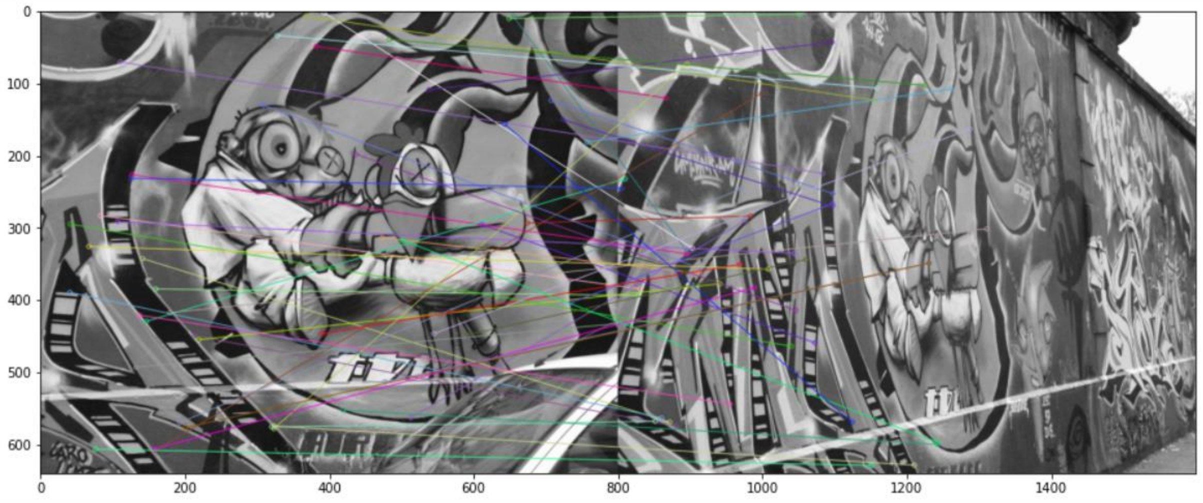
<u>Question 03</u>

In this question we are asked to find and match the SIFT features between img1.ppm and img5.pp. This action can be performed using the SIFT_create function from the Open CV inbuilt functions.

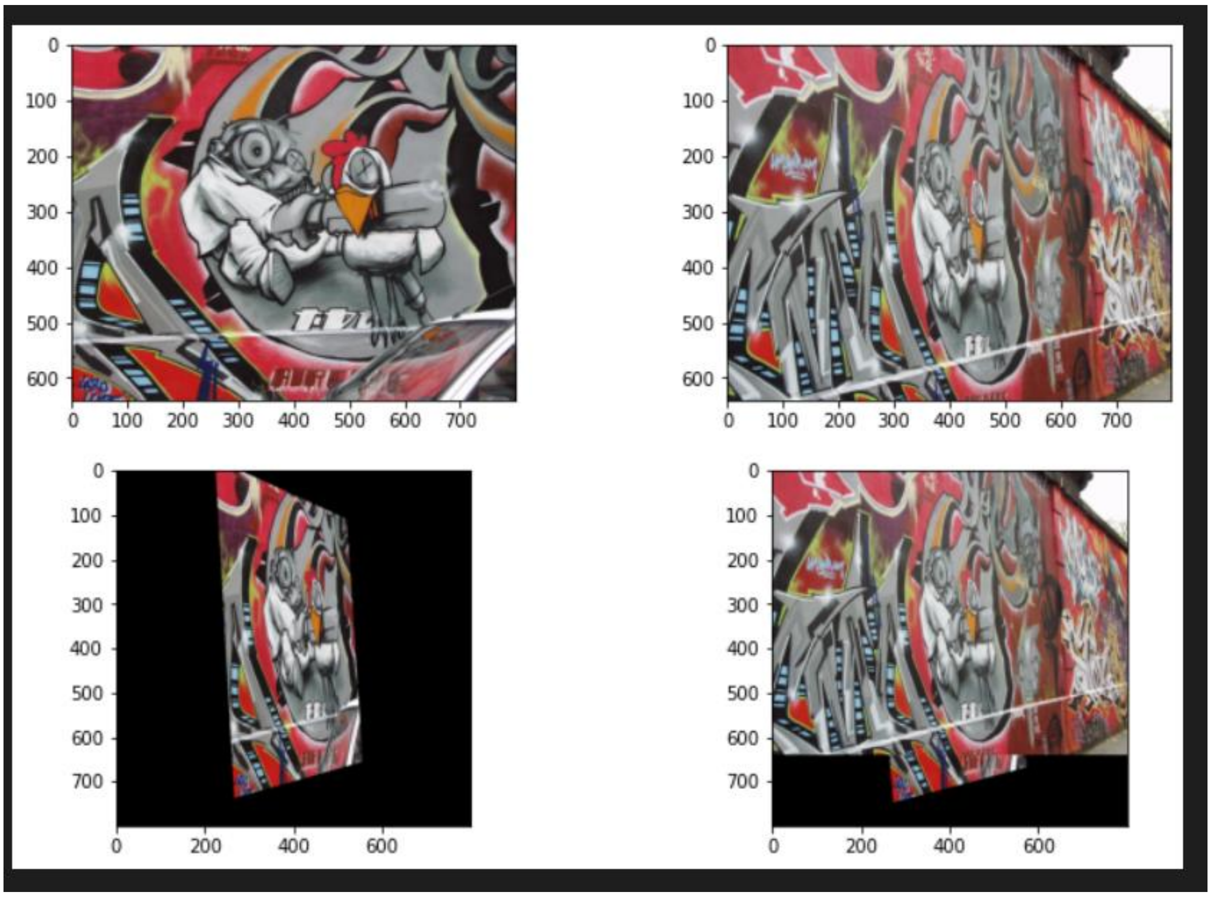


```
#Question 03
img1 = cv.imread(r'img1.ppm')
img1 = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
img5 = cv.imread(r'img5.ppm')
img5 = cv.cvtColor(img5, cv.COLOR_BGR2GRAY)

sift = cv.SIFT_create()
key_1, descriptor_1 = sift.detectAndCompute(img1, None)
key_2, descriptor_2 = sift.detectAndCompute(img5, None)


bf = cv.BFMatcher(cv.NORM_L1, crossCheck=True)
matches = bf.match(descriptor_1,descriptor_2)
matches = sorted(matches, key = lambda x:x.distance)

Matched_img = cv.drawMatches(img1, key_1, img5, key_2, matches[:50], img5, flags=2)
plt.figure(figsize=(16,16))
plt.imshow(Matched_img)
plt.show()
```

The following image shows the matched images but as we can see most of the matches are incorrect and there is a noticeable different in the camera view point of the images. So we can conclude its difficult to get the matches using the function



Stitching the img1 with img 5 was done using the Homography function as well as the warperspective function



GitHub link : https://github.com/Aakkash1999/Image-Processing-02