**EN2550 - Fundamentals of Image Processing & Machine Vision**

**Assignment 2**

**Fitting and Alignment**

**190539T - Sajeepan.T**

**Question-01**

RNASAC algorithm

def getRandomPoints(data, n): #return n random points from data

    count = 0

    sample = []

    while count < n:

        index = np.random.randint(len(data))

        x = data[index][0]

        y = data[index][1]

        if (x, y) not in sample:

            sample.append((x, y))

            count += 1

    return sample

def getCircleParams(sample): #return circle parameters from sample

    pt1 = sample[0]

    pt2 = sample[1]

    pt3 = sample[2]

    A = np.array([[pt2[0] - pt1[0], pt2[1] - pt1[1]], [pt3[0] - pt2[0], pt3[1] - pt2[1]]])

    B = np.array([[pt2[0]\*\*2 - pt1[0]\*\*2 + pt2[1]\*\*2 - pt1[1]\*\*2], [pt3[0]\*\*2 - pt2[0]\*\*2 + pt3[1]\*\*2 - pt2[1]\*\*2]])

    inv\_A = inv(A)

    a, b = np.dot(inv\_A, B) / 2

    a, b = a[0], b[0]

    r = np.sqrt((a - pt1[0])\*\*2 + (b - pt1[1])\*\*2)

    return (a, b, r)

def getInliersAndDistance(circle, data, threshold):

    x\_data = data[:, 0]

    y\_data = data[:, 1]

    a = circle[0]

    b = circle[1]

    r = circle[2]

    total\_distance = 0

    inliers = []

    for i in range(len(x\_data)):

        distance = np.sqrt((x\_data[i] - a)\*\*2 + (y\_data[i] - b)\*\*2)

        if abs(distance - r) <= threshold:

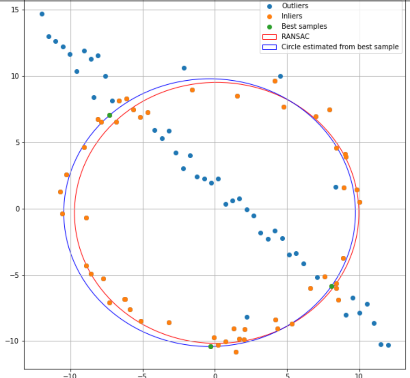
            inliers.append([x\_data[i], y\_data[i]])

            total\_distance += abs(distance - r)

    return inliers, total\_distance

* Initialize the parameters and run the iterations to 35 samples to get an best approximation

Output Results



Question -02

Code:

#function for getting co-ordinates by mouse click

def click\_event(event, x, y, flags, params):

    if event == cv.EVENT\_LBUTTONDOWN:

        print(x, ' ', y)

#function to superimpose

def superimpose(im\_src, im\_dst, pts\_src, pts\_dst):

    h, status = cv.findHomography(pts\_src, pts\_dst)

    im\_out = cv.warpPerspective(im\_src, h, (im\_dst.shape[1], im\_dst.shape[0]))

    return cv.add(im\_out, im\_dst)

* dst1, dst2 are destination images and src1 &src2 are source images

# Four corners of the book in destination image.

pts\_dst1 = np.array([[191, 306], [596, 344], [606, 603], [177, 620]])

pts\_dst2 = np.array([[515, 238], [1022, 156],  [1020, 506], [516, 482]])

# Four corners of the book in source image

pts\_src1 = np.array([[0, 0], [254, 0], [254, 127], [0, 127]])

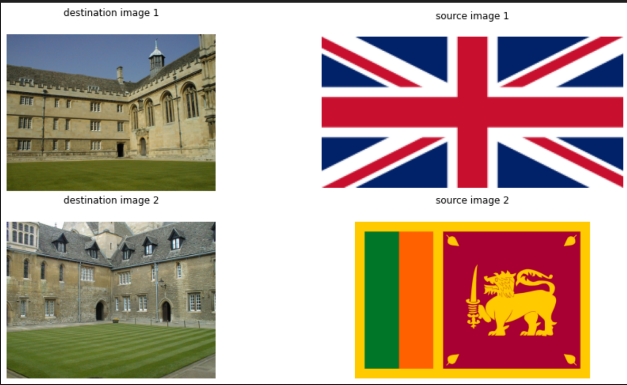
pts\_src2 = np.array([[0, 0], [3499, 0], [3499, 2329], [0, 2329]])

#get the superimposed image

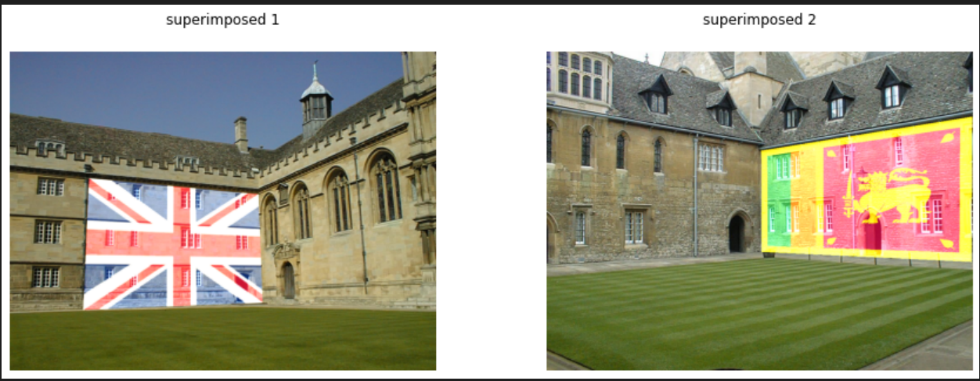
superimposed1 = superimpose(src1, dst1, pts\_src1, pts\_dst1)

superimposed2 = superimpose(src2, dst2, pts\_src2, pts\_dst2)

Input images:



output results



Question -03 Code

# Initiate SIFT detector

def getHomogrpahy(src\_img, dst\_img):

    sift = cv.SIFT\_create()

    # find the keypoints and descriptors with SIFT

    kp1, des1 = sift.detectAndCompute(src\_img, None)

    kp2, des2 = sift.detectAndCompute(dst\_img, None)

    # FLANN parameters

    FLANN\_INDEX\_KDTREE = 1

    index\_params = dict(algorithm = FLANN\_INDEX\_KDTREE, trees = 5)

    search\_params = dict(checks = 50)   # or pass empty dictionary

    flann = cv.FlannBasedMatcher(index\_params, search\_params)

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

    # store all the good matches as per Lowe's ratio test.

    good = []

    for m,n in matches:

        if m.distance < 0.7\*n.distance:

            good.append(m)

    src\_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1,1,2)

    dst\_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1,1,2)

    H, mask = cv.findHomography(src\_pts, dst\_pts, cv.RANSAC, 0.95)

    return H

# Initiate SIFT detector

def matchSIFT(img1, img2):

    sift = cv.SIFT\_create()

    # find the keypoints and descriptors with SIFT

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

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

    # FLANN parameters

    FLANN\_INDEX\_KDTREE = 1

    index\_params = dict(algorithm = FLANN\_INDEX\_KDTREE, trees = 5)

    search\_params = dict(checks = 50)   # or pass empty dictionary

    flann = cv.FlannBasedMatcher(index\_params, search\_params)

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

    # Need to draw only good matches, so create a mask

    matchesMask = [[0,0] for i in range(len(matches))]

    # ratio test as per Lowe's paper

    for i,(m,n) in enumerate(matches):

        if m.distance < 0.7\*n.distance:

            matchesMask[i] = [1,0]

draw\_params = dict(matchColor = (0, 255, 0),

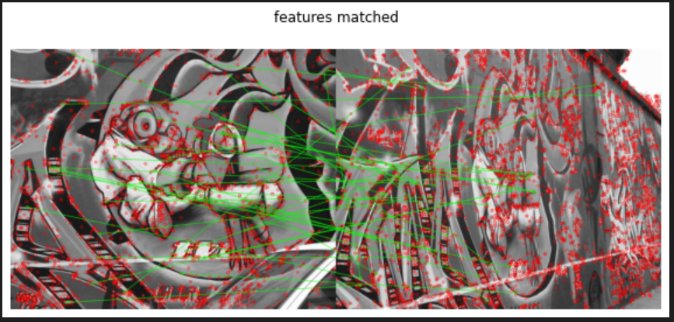
                    singlePointColor = (255, 0, 0),

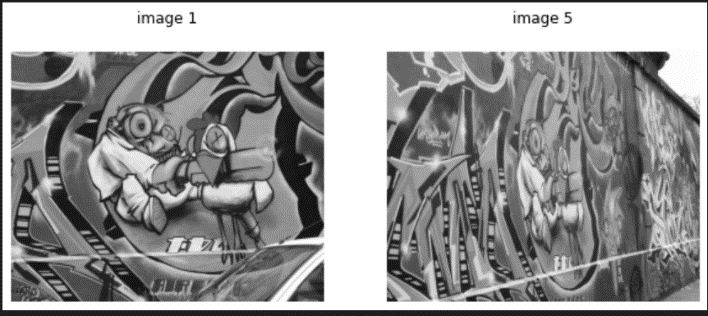
                    matchesMask = matchesMask,

                    flags = cv.DrawMatchesFlags\_DEFAULT)

    img3 = cv.drawMatchesKnn(img1, kp1, img2, kp2, matches, None, \*\*draw\_params)

## view the image

Input images: Output Images:



H = np.identity(3)

for i in range(4):

    src\_name = str(i + 1)+'.ppm'

    src\_img = cv.imread(r'img'+src\_name, cv.IMREAD\_GRAYSCALE)

    dst\_name = str(i + 2)+'.ppm'

    dst\_img = cv.imread(r'img'+dst\_name, cv.IMREAD\_GRAYSCALE)

    H = np.matmul(getHomogrpahy(src\_img, dst\_img), H)

print(H)

Results:

[[ 6.22389620e-01 4.90051516e-02 2.20882134e+02]

[ 2.22416106e-01 1.14261758e+00 -2.34633970e+01]

[ 4.95739539e-04 -6.21289797e-05 9.93217260e-01]]

#wraped image

dst = cv.warpPerspective(src\_img, H, (dst\_img.shape[1] + src\_img.shape[1], dst\_img.shape[0] + src\_img.shape[0]))

#paste them together

dst[0:dst\_img.shape[0], 0:dst\_img.shape[1]] = dst\_img

Output Result:

