Task 1: Image Stitching

Index

List of Figures

1] Overview:

The goal of this task is to stitch two images (named “left.jpg” and “right.jpg”) together to construct a panorama image.

Panoramic photography is a technique that combines multiple images from the same rotating camera to form a single, wide photo. It combines images based on their matching features by the process called image stitching.

Open CV version used is “3.4.2.17”

```
!pip install opencv-python===3.4.2.17
!pip install opencv-contrib-python===3.4.2.17
```

2] Finding Keypoints:

After converting the image to grayscale I have used SIFT and SURF to find the keypoints and descriptors of the given images.

```
left_gray_img = cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY)
right_gray_img = cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY)

surf = cv2.xfeatures2d.SURF_create()
sift = cv2.xfeatures2d.SIFT_create()

key_points1 = sift.detect(gray1, None)
key_points1, des1 = surf.compute(left_gray_img, key_points1)

key_points2 = sift.detect(gray2, None)
key_points2, des2 = surf.compute(right_gray_img, key_points2)
```

Fig 1: Detecting keypoints

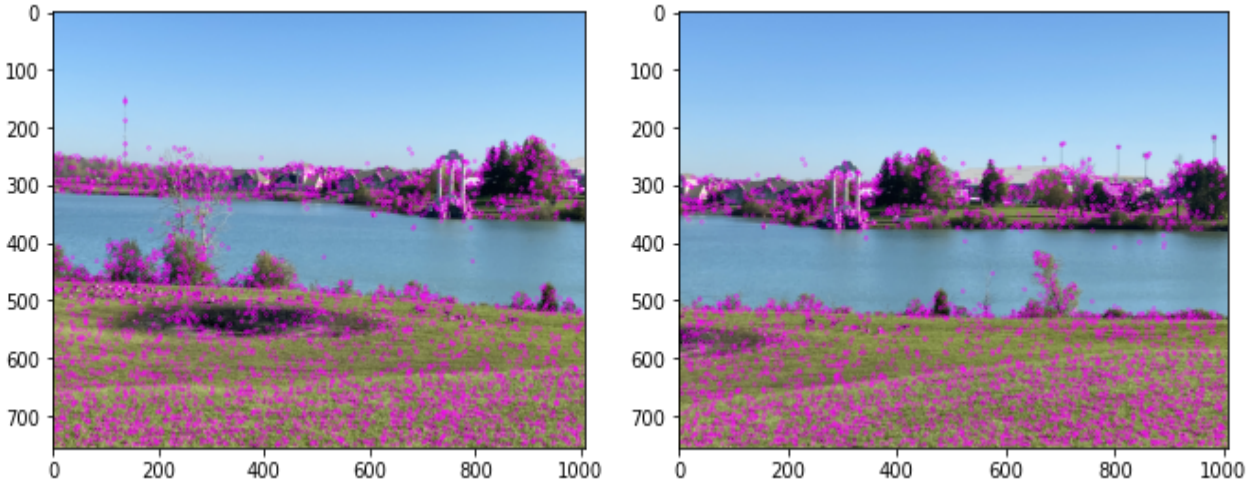

Fig 2: Keypoints Detected [left, right]

The above images show the key points detected in the left and the right image respectively.

3] Matching the points between two images:

Once I extracted the features, I moved on to matching these features between the 2 images.
In order to match the features, I compared the descriptors from the first image with descriptors from the second image. I loop through the 2 descriptors list and find the k nearest neighbors for each query descriptor. That is the nearest neighbor for each descriptor in image 1 is found in image 2 for k=2. This gives me a list of all top 2 matching points between each descriptor of the 2 images. But not all of these points are good matches and contain some outliers.
So my next step was to find only good matches among these matches. For which I set a threshold of 0.75. So all the matching points of image 1 greater than 0.75 times the point in image 2 were considered as good matches. There were 244 good matches found between the 2 images that will be further used for stitching them together.

```
def match_keypoint(kp1, kp2, des1, des2):
    #k-Nearest neighbours between each descriptor
    i = 0
    k=2
    all_matches = []
    for d1 in des1:
      dist = []
      j = 0
      for d2 in des2:
          dist.append([i, j, np.linalg.norm(d1 - d2)])
          j = j + 1
      dist.sort(key=lambda x: x[2])
      all_matches.append(dist[0:k])
      i = i + 1

    # Ratio test to get good matches
    good_matches = []
    for m, n in all_matches:
        if m[2] < 0.75*n[2]:
            left_pt = kp1[m[0]].pt
            right_pt = kp2[m[1]].pt
            good_matches.append(
                [left_pt[0], left_pt[1], right_pt[0], right_pt[1]])
    return good_matches
```

Fig 3:Finding Good Matches

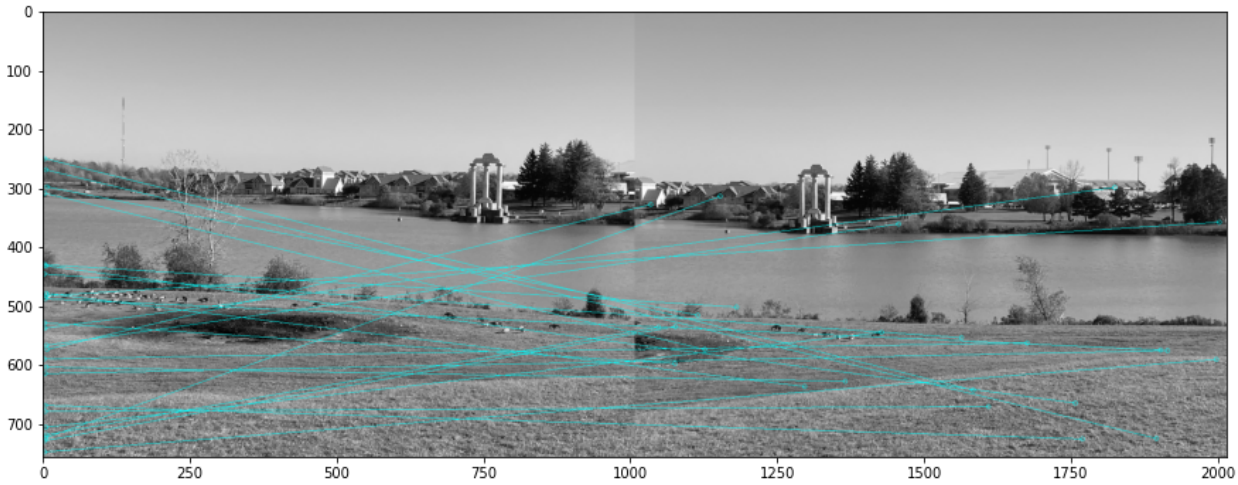

Fig 4: Pairs of Matching Keypoints

4] Computing the homography matrix using RANSAC algorithm:

RANSAC is used to find the commonality between two sets of points for feature-based detection. Homography matrix is required to calculate RANSAC.

2 images in the same scene are relate by homography. It is a transformation that maps the points in one image to the corresponding points in the other image.

Using homography I can take a point $A(x1,y1)$ in the first image and map this point A to the corresponding point $B(x2,y2)$ in the second image.

In the RANSAN function, I have chosen random points to find their homography and compute the best inliers. Running it for 10000 interactions, I compute the homography and get the last row of matrix A using the SVD function. After normalizing it, I use it to find the best inliers, That is the points that have a minimum distance greater than 4.5. By which I get 130 inliers between the 2 images.

```
def compute_homography(points):
    A = []
    for pt in points:
      x, y = pt[0], pt[1]
      X, Y = pt[2], pt[3]
      A.append([x, y, 1, 0, 0, 0, -1 * X * x, -1 * X * y, -1 * X])
      A.append([0, 0, 0, x, y, 1, -1 * Y * x, -1 * Y * y, -1 * Y])

    A = np.array(A)
    u, s, vh = np.linalg.svd(A)
    H = (vh[-1, :].reshape(3, 3))
```

```
    H = H/ H[2, 2]
    return H

def ransac(good_pts):
    best_inliers = []
    final_H = []
    for i in range(5000):
        random_pts = random.choices(good_pts, k=4)
        H = compute_homography(random_pts)
        inliers = []
        t = 5
        for pt in good_pts:
            p = np.array([pt[0], pt[1], 1]).reshape(3, 1)
            p_1 = np.array([pt[2], pt[3], 1]).reshape(3, 1)
            Hp = np.dot(H, p)
            Hp = Hp / Hp[2]
            dist = np.linalg.norm(p_1 - Hp)

            if dist < t: inliers.append(pt)

        if len(inliers) > len(best_inliers):
            best_inliers,final_H = inliers,H
    return final_H
```

Fig 5: Computing Homography using RANSAC

5] Image Warping and Stiching

Homography matrix is used to transform the second image to have the same perspective as the first one which will be kept as the reference frame. Then, I have extracted information about the transformation of the second image and used that information to align the second image with the first one.

- I chose points that are related by homography to change the perspective of the second image using the first image as a reference frame.
- points1 represents coordinates of a reference image, and the second list called temp_points represents coordinates of a second image that is to be transformed.
- cv2.perspectiveTransform() is used to calculate the transformation matrix. And finally warped the second image using the function cv2.warpPerspective()
- Finally stitching is done based on the matches and homography matrix calculated using RANSAC.