Assignment 02 - Fitting and Alignment

Herath H.M.S.I. – Github Profile – 210218M

October 23, 2024

---

1 Question 01 - Blob Detection

Blobs are region of images that is significantly different in properties. Blobs are detected with convolving with laplasian of gaussian kernel and then detecting extremas for different variances.

$$\nabla^2 G(x,y) = \frac{1}{2\pi\sigma^2}\left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right) e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

```
def LoG(s):
    hw = round(3*s)               # For gaussian to be spreaded fully over kernel
    X, Y = np.meshgrid(np.arange(-hw, hw + 1, 1), np.arange(-hw, hw + 1, 1))
    log = ((X**2 + Y**2)/(2*s**2) - 1) * np.exp(-(X**2 + Y**2)/(2*s**2)) / (np.pi * s**4)
    return log * (s**2)           #for normalization
```

Image is convolved with normalized LoG kernel and extremas are detected for corresponding σ. Radius of a blob related to the σ by $\sigma = \frac{r}{\sqrt{2}}$. For detecting extremas scipy maximum_filter function used and threshold of 0.2 used.

```
threshold = 0.2
log_image = cv2.filter2D(gray, -1, LoG(sigma))
local_max = maximum_filter(log_image, size=
    neighborhood_size) == log_image
blobs = (log_image > threshold) & local_max
```

Maximum blob size detected on the threshold 0.1 is around r $= 200 \cdot \sqrt{2} = 282$

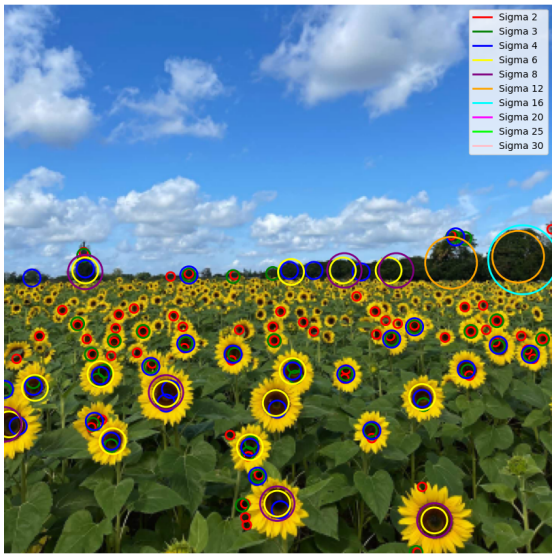

Image with blobs detected

2 Question 02 - RANSAC

RANSAC Algorithm which is highly resistant to the noise is implemented in this section.

```
def point_to_line_dist(params, points):
    a, b, d = params
    norm_ab = np.sqrt(a**2 + b**2)
    dists = np.abs(a * points[:, 0] + b * points[:, 1] + d) / norm_ab
    return dists

# RANSAC algorithm to fit a line to the data
def ransac_fit_line(data_points, max_iters=1000, distance_threshold=0.6, min_inlier_count
    =35):
    optimal_model, optimal_inliers = None, []
    for _ in range(max_iters):
        # Select two random points to define a line
        pt1, pt2 = data_points[np.random.choice(len(data_points), 2, replace=False)]

        # Calculate the line parameters (ax + by + d = 0) using the two points
        a, b, d = pt2[1] - pt1[1], pt1[0] - pt2[0], -(pt2[1] - pt1[1]) * pt1[0] - (pt1[0] -
            pt2[0]) * pt1[1]

        # Normalize to ensure ||[a, b]|| = 1
        norm_factor = np.hypot(a, b)
        a, b, d = a / norm_factor, b / norm_factor, d / norm_factor

        # Compute distances of all points from the line
        dist_from_line = np.abs(a * data_points[:, 0] + b * data_points[:, 1] + d)
        inliers = data_points[dist_from_line < distance_threshold]

        # Update if this model has more inliers
        if len(inliers) > len(optimal_inliers) and len(inliers) >= min_inlier_count:
            optimal_inliers, optimal_model = inliers, [a, b, d]

    return optimal_model, optimal_inliers
```

Fitting the circle first would misclassify points on the line as outliers due to large radial errors, leading to an inaccurate circle model and making it harder to fit the line correctly afterward. But some cases it may fit due to RANSACs high robustness to outliers

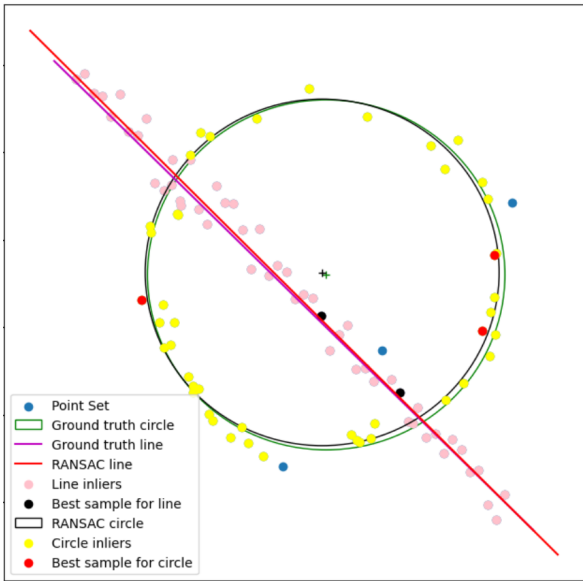

Fig: Fitted Image

3 Question 03 - Image Superimposing

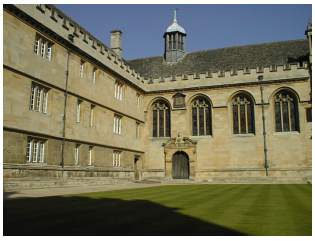

Fig: Image 01

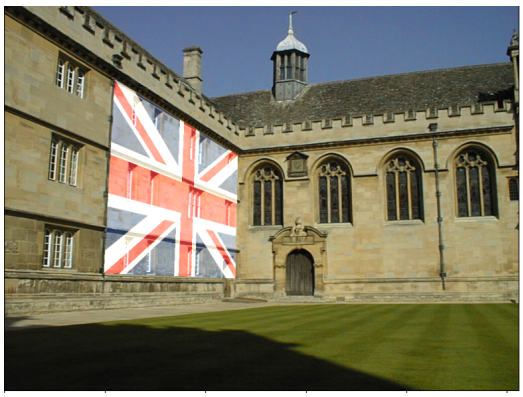

Fig: Superimposed Image

Fig: Image 02

Fig: Image 01

Fig: Superimposed Image

Fig: Image 02

I decided to combine a picture of a green lawn with an image of water to create a scene that looks like the lawn is right next to the sea.

```
# Warped image is im2 and architectural image is im1

homography_matrix, _ = cv2.findHomography(points2, points1)
flag_warped = cv2.warpPerspective(im2, homography_matrix, (im1.shape[1], im1.shape[0]))
result_image = cv2.addWeighted(im1, 1, flag_warped, 0.7, 0)
```

4 Question 04 - Image Allignment

a)

```
#keypoints are computed
k1, des1 = sift.detectAndCompute(img1, None)
k5, des5 = sift.detectAndCompute(img5, None)

matches = bf.knnMatch(des1, des5, k=2)
match_img = cv2.drawMatches(img1, k1, img5, k5, good_matches, None, flags=cv2.
    DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
```

Fig: SIFT Features

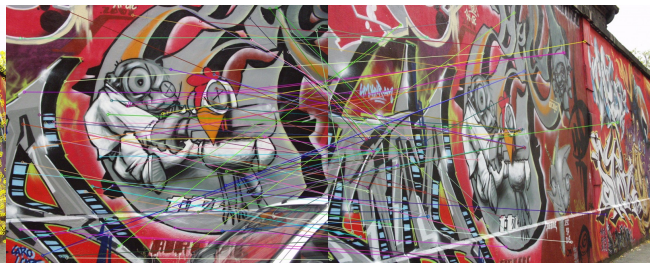

Fig: SIFT Matchings

SIFT features are poorly matched between image 1 and 5 to compute a homography. Therefore, intermediate images, which are more similar to each other are used to compute homography between two images.

b)

```
Computed Homography:
[[ 6.59049226e-01  9.87325525e-01 -1.66065612e+02]
 [-1.26120839e+00  1.44886723e+00  4.50184024e+02]
 [-2.76884468e-03  4.52303892e-03  1.00000000e+00]]

Original Homography :
[[ 6.2544644e-01  5.7759174e-02  2.2201217e+02]
 [ 2.2240536e-01  1.1652147e+00 -2.5605611e+01]
 [ 4.9212545e-04 -3.6542424e-05  1.0000000e+00]]
```

Computed and Original Homography Matrices

Stiched Image

```
def ransac_matching(img1, img2, num_iterations=1000, max_distance=10, min_inliers=100):
    sift, bf = cv2.SIFT_create(), cv2.BFMatcher()
    keypoints1, descriptors1 = sift.detectAndCompute(img1, None)
    keypoints2, descriptors2 = sift.detectAndCompute(img2, None)

    best_matches, best_homography, best_inliers = [], None, []

    for _ in range(num_iterations):
        random_matches = random.sample(range(len(keypoints1)), 4)
        src_pts = np.float32([keypoints1[m].pt for m in random_matches]).reshape(-1, 1, 2)
        dst_pts = np.float32([keypoints2[m].pt for m in random_matches]).reshape(-1, 1, 2)
        homography, _ = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, max_distance)

        if homography is None: continue
        transformed_pts = cv2.perspectiveTransform(src_pts, homography)
        distances = np.sqrt(np.sum((dst_pts - transformed_pts) ** 2, axis=2))
        inliers = np.count_nonzero(distances < max_distance)

    filtered_matches = [cv2.DMatch(i, i, 0) for i, is_inlier in enumerate(best_inliers) if
        is_inlier]
    return filtered_matches, best_homography, keypoints1, keypoints2
```