EN3160 – Image Processing and Machine Vision

Assignment 2 - Fitting and Alignment

Name : A. C. Pasqual
Index No. : 200445V
Submission Date : 4th October 2023
GitHub Repository: https://github.com/Anuki16/EN3160_assignment2

Question 1 – Blob Detection

Considering the relationship $\sigma = r/\sqrt{2}$ to optimally detect a blob with radius r, σ values corresponding to r in range (1, 10) was used to detect blobs.

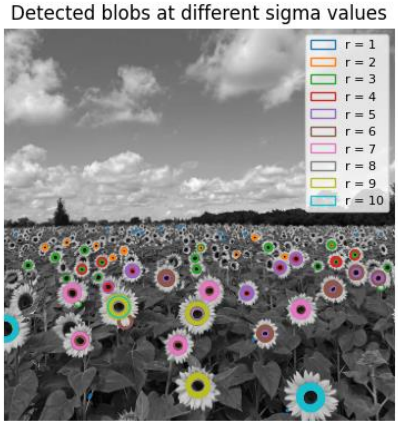
```
def laplace_of_gaussian(sigma):
    hw = round(3*sigma)    # Half width of kernel
    X, Y = np.meshgrid(np.arange(-hw, hw + 1, 1), np.arange(-hw, hw + 1, 1))
    log = ((X**2+Y**2)/(2*sigma**2)-1) * np.exp(-(X**2+Y**2)/(2*sigma**2))/(np.pi * sigma**4)
    return log

def detect_max(img_log, sigma):
    coordinates = []
    (h, w) = img_log.shape
    k = 1
    for i in range(k, h-k):
        for j in range(k, w-k):
            slice_img = img_log[i-k:i+k+1, j-k:j+k+1]
            result = np.max(slice_img) # finding maximum
            if result >= 0.09: # threshold
                x, y = np.unravel_index(slice_img.argmax(), slice_img.shape)
                coordinates.append((i+x-k, j+y-k)) #finding co-rdinates
    return set(coordinates)
```

```
# Finding local peaks for each sigma
sigma = r/1.414
LOG = sigma**2 * laplace_of_gaussian(sigma)
img1_log = np.square(cv.filter2D(img1, -1, LOG))
coordinates = detect_max(img1_log, sigma)
```

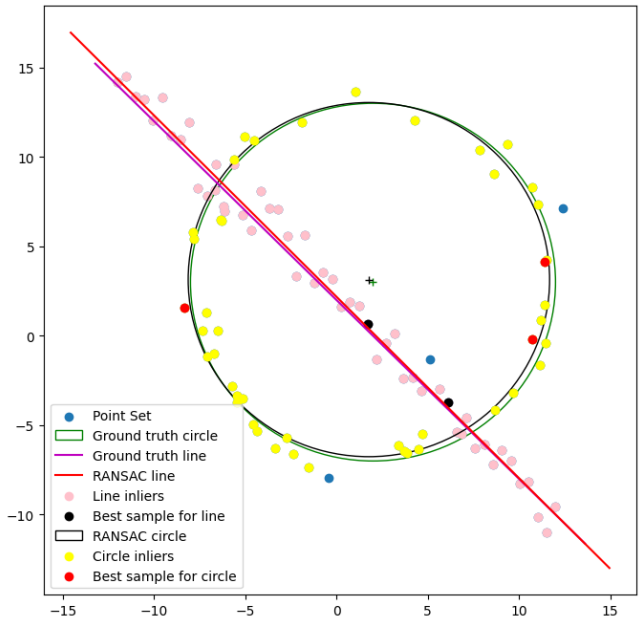
Original Image

Detected blobs at different sigma values



This image shows the r value that was able to detect each blob. The largest circles were detected at r = 9 and 10.

Question 2 – Line and circle fitting using RANSAC



Parameters of RANSAC used:

s = 2 for line and s = 3 for circle
They are the minimum number of points required to uniquely define the shape.

Error threshold t = 1 for line, t = 1.2 for circle
Since the nature of noise of the points is unknown, the optimal value for t was found through trial and error.

Consensus size d = 40 for both line and circle
In the original point set there are 50 points each from the line and circle, therefore d = 40 captures a sufficient number of inliers.

```
# Fitting the line using RANSAC (procedure is similar for circle)
for i in range(iters):
    indices = np.random.choice(np.arange(0, N), size=min_points, replace=False)
    params = line_eq(X[indices[0]], X[indices[1]])
    inliers = consensus_line(params, thres, X)[0]
    if len(inliers) >= d:   # compute again
        res = least_squares_line_fit(inliers, params, X)
        if res.fun < best_error:
            best_error = res.fun
            best_model_line = params
            best_fitted_line = res.x
            best_line_inliers = inliers
            best_sample_points = indices

line_inliers = consensus_line(best_fitted_line, 1.2, X)[0]
```

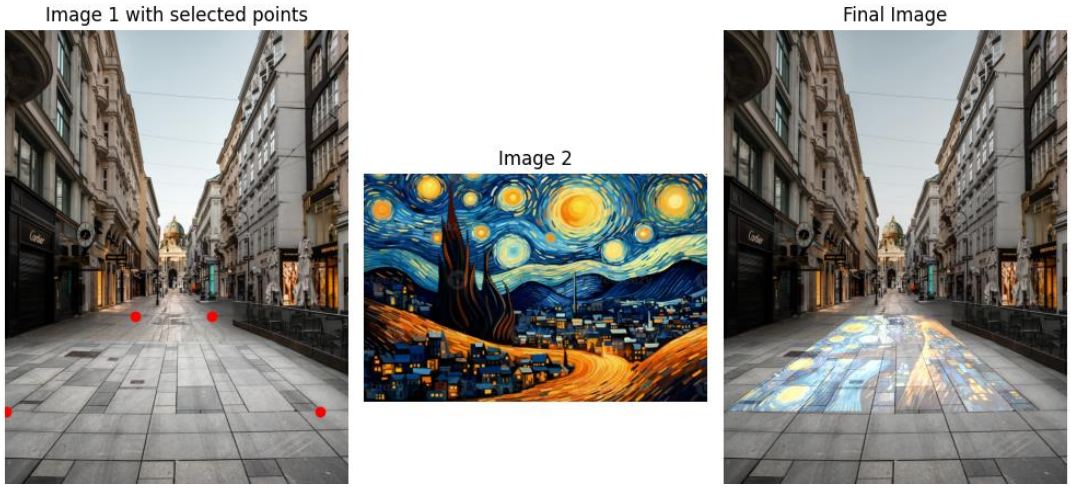
```
def least_squares_line_fit(indices, initial, X):  # line fitting with scipy minimize
    res = minimize(fun=tls_error_line, x0=initial, args=(indices, X),
constraints=constraint_dict, tol=1e-6)
    return res
```

```
# Squared error calculation for line and circle
def tls_error_line(params, indices, X):
    a, b, d = params
    error = np.sum((a * X[indices, 0] + b * X[indices, 1] - d)**2)
    return error

def tls_error_circle(params, indices, X):
    cx, cy, r = params
    error = np.sum((dist((cx, cy), (X[indices, 0], X[indices, 1])) - r)**2)
    return error
```

If the circle is fitted first, there is a chance of the three random sample points all being on the line. Then the calculated circle will be large and locally similar to a line. However, since the RANSAC algorithm will run for many iterations with different sample points, it is still possible to accurately fit the circle without removing the line points.

Question 3 – Superimposing an image on another




```
def superimpose(image, logo, dst_points, beta = 0.3, alpha = 1):

    y, x, _ = logo.shape
    src_points = [(0, y), (x, y), (x, 0), (0, 0)]   # bl, br, tr, tl corners
    src_points = np.array([np.array(p) for p in src_points])

    # Compute the homography
    tform = transform.estimate_transform('projective', src_points, dst_points)
    tf_img = transform.warp(logo, tform.inverse)
    tf_img = (tf_img * 255).astype(np.uint8)

    # Weighted addition to blend the two images
    dst = cv.addWeighted(img3, alpha, tf_img, beta, 0)
    return dst
```

In some cases, padding was added to image 2 to match the dimensions of image 1 when combining.

Question 4 – Image stitching

Image 1 SIFT features

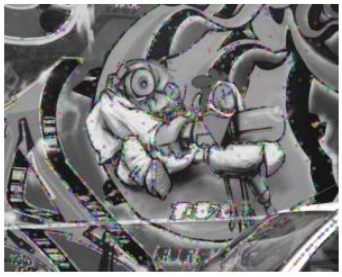


Image 5 SIFT features

Matched features between image 1 and 5

There are not enough matching features between images 1 and 5 for a homography to be accurately computed. Therefore, homographies were computed between images 1-2, 2-3 etc. which are more similar to each other and image 1 was passed through all of the transforms to map it to image 5.

Image 1

Image 5

Transformed Image 1

Stitched Image

```
Computed Homography
[[ 6.11404459e-01  5.03189502e-02  2.21391678e+02]
 [ 2.11980223e-01  1.14096503e+00 -2.14952739e+01]
 [ 4.74861344e-04 -5.18621014e-05  9.90604809e-01]]
```

```
Actual 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]]
```

The computed and actual homographies are very similar. (Sum of squared errors = 17.28)

```python
# RANSAC for finding best homography
for i in range(200):
    chosen_matches = np.random.choice(good_matches, 4, replace = False)
    src_points = []; dst_points = []
    for match in chosen_matches:
        src_points.append(np.array(keypoints1[match.queryIdx].pt))
        dst_points.append(np.array(keypoints5[match.trainIdx].pt))

    tform = transform.estimate_transform('projective', src_points, dst_points)
    inliers = get_inliers(src_full, dst_full, tform, thres)
    if len(inliers) > best_inlier_count:
        best_homography = tform
```

```python
final_transform = transform.ProjectiveTransform(np.identity(3))
for i in range(1, 5):  # Computing adjacent transforms
    img1, img2 = images[i-1], images[i]
    good_matches, keypoints1, keypoints5 = SIFT_features(img1, img2)
    tform, _ = find_best_homography(good_matches, keypoints1, keypoints5)
    final_transform = final_transform + tform   # Combining the transforms
final_transformed_image = transform.warp(images[0], final_transform.inverse)
```