EN3160 IMAGE PROCESSING AND MACHINE VISION

Assignment 2: Fitting and Alignment

De Zoysa A S I (220106D)

Link to GitHub repository - https://github.com/Sithum02/EN3160-Assignments

Question 1

The Code

```python
# Detect maxima manually
    coordinates = []
    (H, W) = img_log.shape
    k = 1  # Neighborhood size
    threshold = 0.1  # Threshold for maxima detection
    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]
            if np.max(slice_img) >= threshold:
                # Get coordinates of the maximum point
                x, y = np.unravel_index(slice_img.argmax(), slice_img.shape)
                coordinates.append((i + x - k, j + y - k))
    coordinates = set(coordinates)

    # Plot circles around detected blobs
    for x, y in coordinates:
        c = plt.Circle((y, x), sigma * 1.414, color='g', linewidth=1, fill=False)
        ax.add_patch(c)
    patches.append(c)
    ax.plot()
```

Listing 1: Blob detection code.

Figure 1: Blob detection

Radius Range Integers from 1 to 10.

σ **Values** 0.71, 1.41, 2.12, 2.83, 3.54, 4.24, 4.95, 5.66, 6.36, 7.07.

Largest Circle $\sigma = 7.07$ and Radius $= 10$.

Question 2

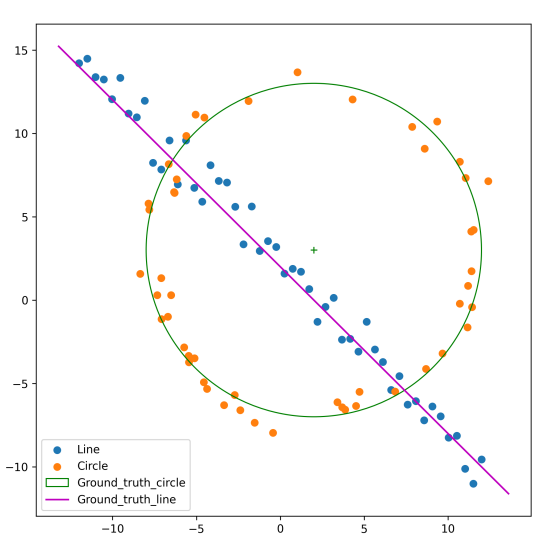
The Code

```python
def ransac_line_fit(points, max_iter=1000, eps=0.6, min_support=35):
    best_model = None
    best_inliers = np.empty((0,2))
    n_pts = len(points)

    for _ in range(max_iter):
        idx = np.random.choice(n_pts, 2, replace=False)
        line_params = fit_line_from_two_pts(points[idx[0]], points[idx[1]])
        dist = compute_point_line_dist(line_params, points)
        mask = dist < eps
        if mask.sum() > len(best_inliers) and mask.sum() >= min_support:
            best_model = line_params
            best_inliers = points[mask]
    return best_model, best_inliers

def ransac_circle_fit(points, max_iter=1000, eps=1.5, min_support=45):
    best_model = None
    best_inliers = np.empty((0,2))
    n_pts = len(points)

    for _ in range(max_iter):
        idx = np.random.choice(n_pts, 3, replace=False)
        try:
            circ_params = estimate_circle_from_pts(points[idx])
        except np.linalg.LinAlgError:
            continue
        dist = radial_error(circ_params, points)
        mask = dist < eps
        if mask.sum() > len(best_inliers) and mask.sum() >= min_support:
            best_model = circ_params
            best_inliers = points[mask]
    return best_model, best_inliers

# Fit line then circle on residual points
line_params, line_inliers = ransac_line_fit(X)

# Remove line inliers
mask = np.ones(len(X), dtype=bool)
for pt in line_inliers:
    mask[np.all(np.isclose(X, pt), axis=1)] = False
remaining = X[mask]

circle_params, circle_inliers = ransac_circle_fit(remaining)
```

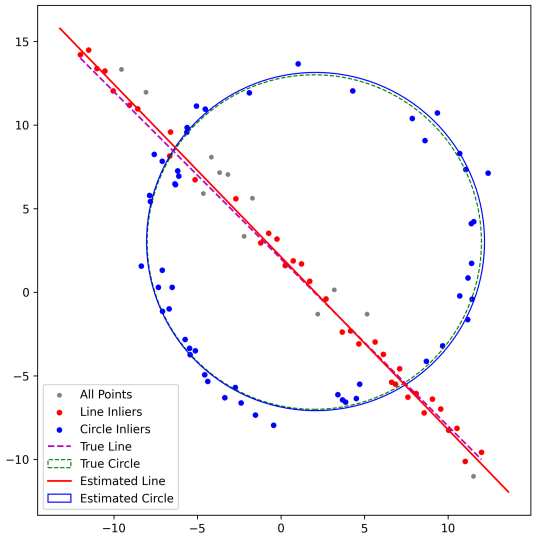
Listing 2: RANSAC code.

RANSAC line fitting parameters are iterations = 1000, threshold = 0.6, and minimum inliers = 35. The parameters of the estimated line are a = -0.72, b = -0.70, d = 1.48. RANSAC circle fitting parameters are iterations = 1000, threshold = 1.5, and minimum inliers = 45. The parameters of the estimated circle are x = 2.07, y = 3.03, r = 10.11.

If we fit the circle first, many line points may be incorrectly classified as circle inliers, leading to an incorrect circle model. As a result, when we subsequently try to fit the line, we are left with a corrupted or reduced set of line points, causing poor line estimation. Therefore, fitting the line first and then the circle gives more robust and accurate results.



(a) Generated Data points



(b) Predictions

Question 3

The Code

```python
# 4 Points were selected by Mouse Clicks

# Define source and destination points for homography
destination_points = np.array(clicked_points).astype(np.float32)
h2, w2 = adding_img.shape[:2]
source_points = np.float32([[0, 0], [w2, 0], [0, h2], [w2, h2]])

# Define the homographic matrix
H = cv.getPerspectiveTransform(source_points, destination_points)
# Warping the adding image with the homography
warped_img = cv.warpPerspective(adding_img, H, (base_img.shape[1], base_img.shape[0]))
# Superimposing the warped image with the base image
supimp_img = cv.addWeighted(base_img, 1, warped_img, 1, 0)
```

Listing 3: Blob detection code.

(a)

(b)

(c)

Figure 3: Overlaying the flag image (b) onto image (a) using Homography. The result is (c).

(a)

(b)

(c)

Figure 4: Overlaying the image (b) onto image (a) using Homography. The result is (c).

Question 4

The Code for (a)

```
# Convert to grayscale
img1_gray = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
img2_gray = cv2.cvtColor(img2, cv2.COLOR_RGB2GRAY)

# Get SIFT descriptors
sift = cv2.SIFT_create(nOctaveLayers=3, contrastThreshold=0.09,
                       edgeThreshold=25, sigma=1)
keypoints1, descriptors1 = sift.detectAndCompute(img1_gray, None)
keypoints2, descriptors2 = sift.detectAndCompute(img2_gray, None)

# Match using Brute Force Matcher
bf = cv2.BFMatcher()
matches = bf.knnMatch(descriptors1, descriptors2, k=2)

# Lowe's ratio to filter best matches
best_matches = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
        best_matches.append(m)
```

Listing 4: SIFT feature matching.

The Code for (b)

```
# RANSAC parameters
num_points = 4
thres = 1.0
iters = 200
```

```python
best_homography = None
best_inlier_count = 0
best_inliers = None

for i in range(iters):
    # Get random matches
    chosen_matches = np.random.choice(good_matches, num_points, 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(keypoints2[match.trainIdx].pt))

    src_points = np.array(src_points)
    dst_points = np.array(dst_points)

    # Estimate homography
    tform = transform.estimate_transform('projective', src_points, dst_points)

    # Get number of inliers (assuming get_inliers function is defined)
    inliers = get_inliers(src_full, dst_full, tform, thres)

    # Check if this is the best model so far
    if len(inliers) > best_inlier_count:
        best_inlier_count = len(inliers)
        best_homography = tform
        best_inliers = inliers
```

Listing 5: RANSAC for homography estimation.

(a) Homography (b) Stitched image

Figure 5: Homography Transformation and stitched image.