Electronic and Telecommunication engineering

University of Moratuwa

EN3160 - Image Processing and Machine Vision

A02 Fitting and Alignment

RANAVIRAJA R.W.H.M.T.A. - Index No. 200520X

GitHub: GitHub

04th October, 2023

Q1.

```
for sigma in sigma_values:
    # Apply LoG (Laplacian of Gaussian) filter
    blurred = cv2.GaussianBlur(gray_image, (0, 0), sigma)
    laplacian = cv2.Laplacian(blurred, cv2.CV_64F)
    abs_laplacian = np.abs(laplacian) # Calculate the absolute Laplacian values
    blob_mask = abs_laplacian > threshold * abs_laplacian.max()  # Create a mask for blobs
    contours, _ = cv2.findContours(blob_mask.astype(np.uint8), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) # Find contours around blobs

    for contour in contours:
        if len(contour) >= 5:
            (x, y), radius = cv2.minEnclosingCircle(contour)
            center = (int(x), int(y))
            radius = int(radius)
            circles.append((center, radius, sigma))
```

Detected Circles

```
Parameters of the largest circle:
Center: (110, 258)
Radius: 15
Sigma value: 2.0
```

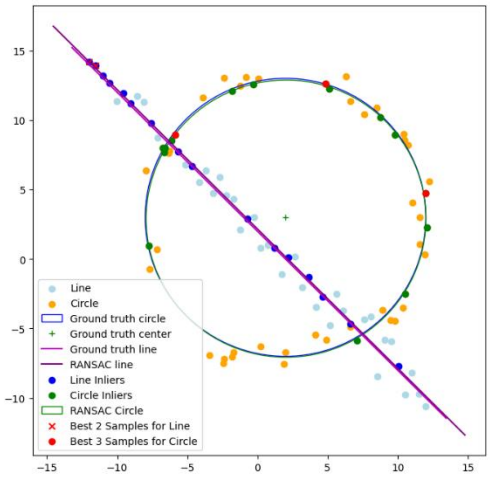
Q2. (a)

```
# RANSAC to fit a line with unit normal constraint
def ransac_line(X, iterations, threshold, min_inliers):
    best_model = None
    best_inliers = []

    for _ in range(iterations):
        sample_indices = np.random.choice(len(X), 2, replace=False)
        x1, y1 = X[sample_indices[0]]
        x2, y2 = X[sample_indices[1]]

        a, b, d = line_equation_from_points(x1, y1, x2, y2)

        # Constraint: Ensure unit normal vector
        magnitude = np.sqrt(a**2 + b**2)
        a /= magnitude
        b /= magnitude

        # Calculate the distance of all points to the line
        distances = np.abs(a*X[:,0] + b*X[:,1] - d)

        # Find inliers based on the threshold
        inliers = np.where(distances < threshold)[0]

        if len(inliers) >= min_inliers:
            if len(inliers) > len(best_inliers):
                best_model = (a, b, d)
                best_inliers = inliers

    return best_model, best_inliers

# RANSAC parameters
iterations = 10000
threshold = 0.15
min_inliers = 15
```

(b)

```
# RANSAC to fit a circle
def ransac_circle(X, iterations, threshold, min_inliers):
    best_model = None
    best_inliers = []

    for _ in range(iterations):
        sample_indices = np.random.choice(len(X), 3, replace=False)
        x1, y1 = X[sample_indices[0]]
        x2, y2 = X[sample_indices[1]]
        x3, y3 = X[sample_indices[2]]

        x_center, y_center, radius = circle_equation_from_points(x1, y1, x2, y2, x3, y3)

        # Calculate the radial error of all points to the circle
        errors = np.abs(np.sqrt((X[:, 0] - x_center)**2 + (X[:, 1] - y_center)**2) - radius)

        # Find inliers based on the threshold
        inliers = np.where(errors < threshold)[0]

        if len(inliers) >= min_inliers:
            if len(inliers) > len(best_inliers):
                best_model = (x_center, y_center, radius)
                best_inliers = inliers

    return best_model, best_inliers

# RANSAC parameters for circle estimation
circle_iterations = 10000
circle_threshold = 0.2  # Adjust the threshold as needed
circle_min_inliers = 15
```

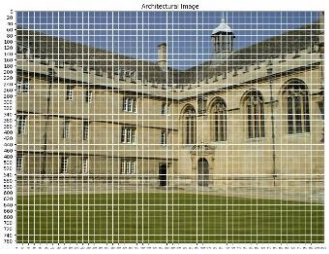
(c)

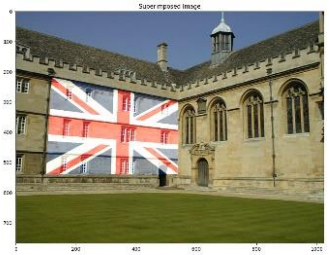


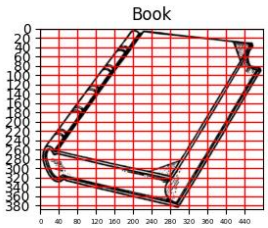
(d) If we fit the circle first, it uses some points which belong to the line. We get a less accurate line after we fit the circle. Therefore, the line should be fit before the circle.

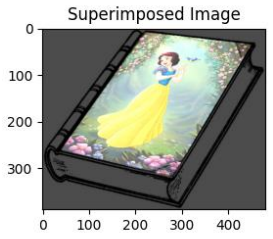
Q3.

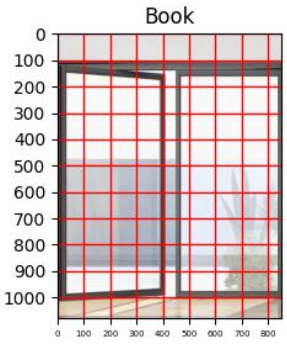
```python
# Define four points on the base image (you should adjust these points)
points_base = np.array([[120, 220], [540, 300], [540, 540], [100, 540]], dtype=np.float32)

# Define the corresponding points on the flag image (keep the same order as base points)
points_flag = np.array([[0, 0], [flag_image.shape[1], 0], [flag_image.shape[1], flag_image.shape[0]], [0, flag_image.shape[0]]], dtype=np.float32)

# Calculate the homography matrix
homography_matrix, _ = cv2.findHomography(points_flag, points_base)

# Warp the flag image using the homography matrix
flag_warped = cv2.warpPerspective(flag_image, homography_matrix, (base_image.shape[1], base_image.shape[0]))

# Blend the warped flag image with the base image
result = cv2.addWeighted(base_image, 1, flag_warped, 0.7, 0)
```

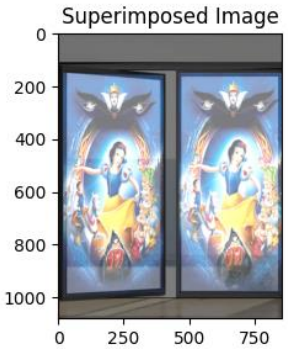






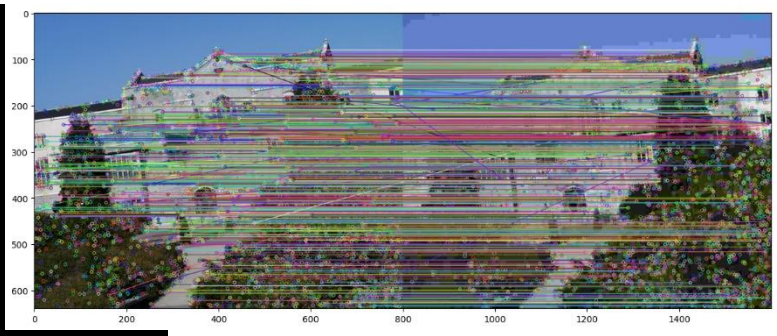






Q4. (a)

```python
# Create a SIFT object
sift = cv2.SIFT_create()

# Find the keypoints and descriptors for both images
keypoints1, descriptors1 = sift.detectAndCompute(img1, None)
keypoints5, descriptors5 = sift.detectAndCompute(img5, None)

# Create a BFMatcher (Brute-Force Matcher) with default parameters
bf = cv2.BFMatcher()

# Match descriptors using KNN (k-nearest neighbors) with k=2
matches = bf.knnMatch(descriptors1, descriptors5, k=2)

# Apply ratio test to filter good matches
good_matches = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
        good_matches.append(m)

# Draw the matches
matching_result = cv2.drawMatches(img1, keypoints1, img5, keypoints5, good_matches, None)

plt.figure(figsize=(15,15))
plt.imshow(cv2.cvtColor(matching_result, cv2.COLOR_BGR2RGB))
```



(b)

```python
# Extract the matched keypoints
src_pts = np.float32([keypoints1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
dst_pts = np.float32([keypoints5[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)

# Use RANSAC to estimate the homography matrix
homography_matrix, _ = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

# Compare the computed homography with the one provided in the dataset
dataset_homography = np.array([[0.877, 0.479, 73.0], [-0.479, 0.877, 320.0], [0.0, 0.0, 1.0]])

# Print and compare the matrices
print("Computed Homography Matrix:")
print(homography_matrix)

print("\nDataset Homography Matrix:")
print(dataset_homography)
```

```
Computed Homography Matrix:
[[ 1.00099597e+00 -6.85431880e-04  3.13209320e-02]
 [ 3.54464572e-04  1.00014375e+00 -1.10532496e-01]
 [ 1.53690428e-06 -8.95294295e-07  1.00000000e+00]]

Dataset Homography Matrix:
[[  0.877   0.479  73.   ]
 [ -0.479   0.877 320.   ]
 [  0.      0.      1.   ]]
```

(c)

```python
# Warp `img1.ppm` using the computed homography to align it with `img5.ppm`
stitched_image = cv2.warpPerspective(img1, homography_matrix, (img5.shape[1], img5.shape[0]))

# Overlay `img1.ppm` onto `img5.ppm`
result_image = cv2.addWeighted(stitched_image, 1, img5, 1, 0)

plt.figure(figsize=(5,5))
plt.imshow(cv2.cvtColor(result_image, cv2.COLOR_BGR2RGB))
plt.show()
```