Electronic and Telecommunication Engineering
University of Moratuwa
EN3160 : Image processing and Machine Vision
Assignment 2 - Fitting

210498T : Priyankan V.

October 23, 2024

1 Question 1

Blob detection identifies regions or objects in an image that stand out due to shared characteristics like color, intensity, or texture.
The Laplacian of Gaussians (LoG) enhances features and detects edges in two steps. First, a Gaussian filter smooths the image, reducing noise and highlighting key structures. Then, the Laplacian operator identifies areas of intensity change. By adjusting σ and threshold values, blobs can be accurately detected, with blobs often resembling ripples from the LoG process.

```python
def generate_gaussian_kernel(dim, sigma_val, display=False):
    """Generate a 2D Gaussian kernel with a specified size and sigma."""
    axis_1D = np.linspace(-(dim // 2), dim // 2, dim)
    kernel_1D = np.exp(-(axis_1D ** 2) / (2 * sigma_val ** 2)) / (sigma_val * np.sqrt(2 * np.pi))
    kernel_2D = np.outer(kernel_1D, kernel_1D)
    kernel_2D /= kernel_2D.max()  # Normalize the kernel so its max value is 1.
    if display:
        plt.imshow(kernel_2D, cmap='gray', interpolation='none')
        plt.title("Gaussian Kernel")
        plt.show()
    return kernel_2D

sigma_val = 0.638
gaussian_kernel_2D = generate_gaussian_kernel(5, sigma_val, display=False)
laplacian_of_gaussian_kernel = cv.Laplacian(gaussian_kernel_2D, cv.CV_64F) * sigma_val**2
laplacian_filtered_image = cv.filter2D(img1, -1, laplacian_of_gaussian_kernel)
threshold = 75 #Adjust
_, binary_output = cv.threshold(np.abs(laplacian_filtered_image), threshold, 255, cv.THRESH_BINARY)
contour_list, _ = cv.findContours(binary_output.astype(np.uint8), cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
final_image = cv.cvtColor(img1, cv.COLOR_GRAY2BGR)

for contour in contour_list:
    (center_point, radius) = cv.minEnclosingCircle(contour)
    center_coords = (int(center_point[0]), int(center_point[1]))
    radius = int(radius)
    cv.circle(final_image, center_coords, radius, (255, 0, 255), 2)  # Purple circles, thickness=2
```

Listing 1: Blob Detection

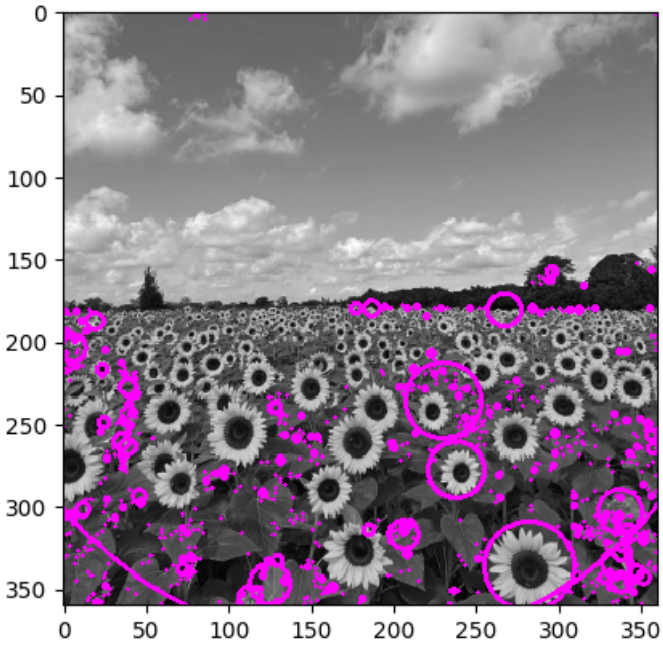

Figure 1: Detected blobs

- Maximum radius got : 80
- Range of σ values : 0.5, 0.6, 0.75, 1.0, 2.0, 3.0
- Selected σ value : 0.64

2 Question 2

To estimate both a line and a circle from a noisy dataset using the RANSAC algorithm, which is an iterative method for estimating the parameters of a mathematical model from observed data that contains outliers.

```
def tls_line(params, indices, X):
    a, b, c = params[0], params[1], params[2]
    residuals = a * X[indices, 0] + b * X[
        indices, 1] - c
    return np.sum(np.square(residuals))

def line_consensus(X, params, threshold):
    a, b, c = params[0], params[1], params[2]
    errors = np.abs(a * X[:, 0] + b * X[:, 1]
         - c)
    return errors < threshold

def circle_constraint(params):
    return params[0]**2 + params[1]**2 - 1

cons = ({'type': 'eq', 'fun':
    circle_constraint})

def circle_distance(params, X, indices):
    xc, yc, radius = params
    squared_differences = np.square(np.sqrt((
        X[indices, 0] - xc)**2 + (X[indices,
        1] - yc)**2) - radius)
    return np.sum(squared_differences)

def circle_consensus(X, params, threshold):
    xc, yc, radius = params
    distances = np.sqrt((X[:, 0] - xc)**2 + (
        X[:, 1] - yc)**2)
    errors = np.abs(distances - radius)
    return errors < threshold
```

Listing 2: RANSAC Line and Circle Fitting

```
while iterations < max_iterations:
    selected_indices = np.random.randint(0, N
        , sample_size)
    initial_guess = np.array([1, 1, 0])
    result = minimize(fun=line_tls, args=(
        selected_indices, X_), x0=
        initial_guess, tol=1e-6, constraints=
        cons)
    current_inliers = consensus_line(X_,
        result.x, tolerance)
    if current_inliers.sum() > min_inliers:
        refined_guess = result.x
        refined_result = minimize(fun=
            line_tls, args=(current_inliers,
            X_), x0=refined_guess, tol=1e-6,
            constraints=cons)
        if refined_result.fun <
            smallest_error:
            best_line_model = refined_result.
                x
            best_sample = X_[selected_indices
                , :]
    iterations += 1
```

Listing 3: Line Fitting Iteration

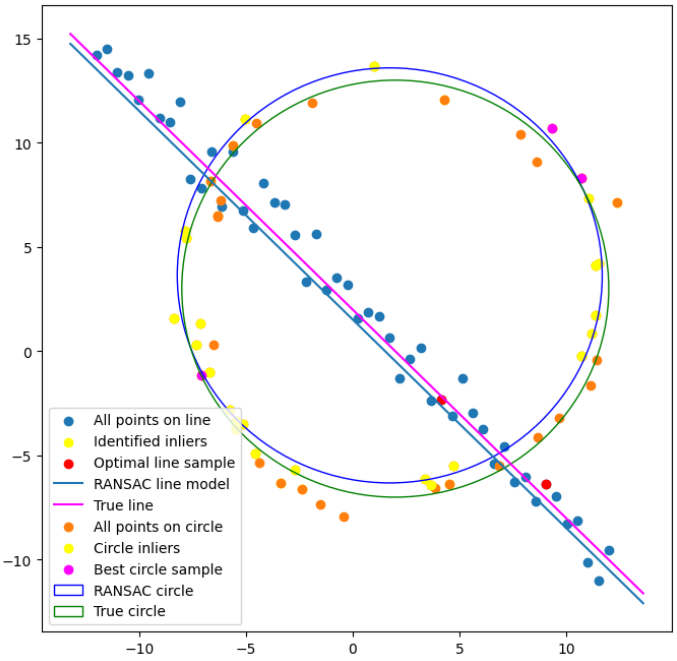

Figure 2: Line and Circle Fitting

3 Question 3

Wrapping one figure and superimposing it to another image.

```
def draw_circle(event,x,y,flags,param):
    global n
    architectural_points = param[0]
    if event == cv.EVENT_LBUTTONDOWN:
        cv.circle(param[1],(x,y),5,(255,0,0)
            ,-1)
        architectural_points[n] = (x,y)
        n += 1
# Getting the mouse points of the base image
cv.namedWindow('Image', cv.WINDOW_AUTOSIZE)
param = [architectural_points,
    architectural_image]
cv.setMouseCallback('Image',draw_circle,
    param)
while(1):
    cv.imshow('Image', architectural_image)
    if n == number_of_points:
        break
    if cv.waitKey(20) & 0xFF == 27:
        break

flag_points = 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)
homography_matrix, _ = cv.findHomography(
    flag_points, architectural_points)
flag_warped = cv.warpPerspective(flag_image,
    homography_matrix, (architectural_image.
    shape[1], architectural_image.shape[0]))
blended_image = cv.addWeighted(
    architectural_image, 1, flag_warped, 0.7,
    0)
```

Listing 4: Superimposing

Figure 3: Fort

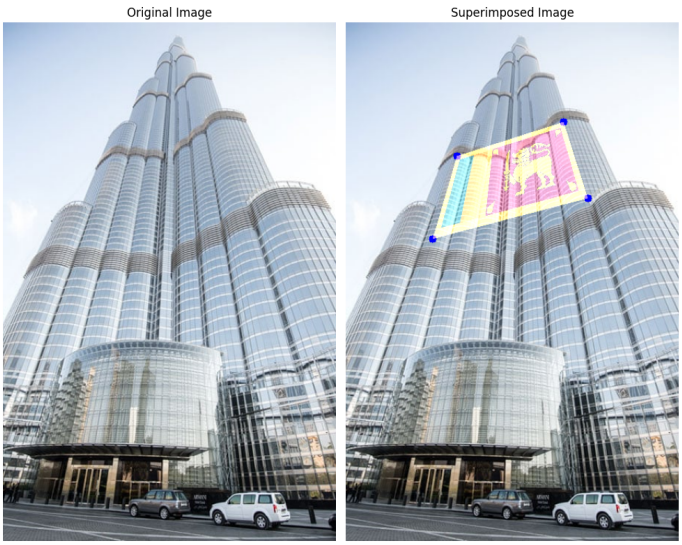

Figure 4: Burj Khalifa

4 Question 4

To stitch two images together, we first detect keypoints and generate descriptors using the SIFT algorithm. These descriptors are matched using a brute-force matcher (BFMatcher). Then, we compute a homography matrix to align one image with the other, using RANSAC to filter outliers. Finally, we warp one image based on the homography to blend it with the other.

(a)

```
# Initialize SIFT detector
sift = cv.SIFT_create(nOctaveLayers=3, contrastThreshold=0.09, edgeThreshold=25, sigma=1)

# Detect keypoints and compute descriptors
kp1, desc1 = sift.detectAndCompute(img1, None)
kp5, desc5 = sift.detectAndCompute(img5, None)

# Match features using BFMatcher and apply ratio test
matches = cv.BFMatcher().knnMatch(desc1, desc5, k=2)
good_matches = [m for m, n in matches if m.distance < 0.75 * n.distance]
```

Listing 5: Creating key points and descriptors

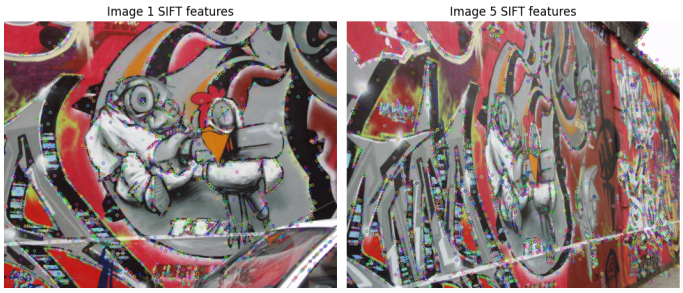

Figure 5: Key Points

Keypoints are mathched using BFmatcher.

Figure 6: Matched Key Points

(b) Homography matrix was found using RANSAC. The following matrix generated as homography matrix.

$$\begin{bmatrix} 6.5608 \times 10^{-1} & -3.9043 \times 10^{-1} & 2.9691 \times 10^{2} \\ 5.8070 \times 10^{-1} & 7.8027 \times 10^{-1} & -1.2334 \times 10^{2} \\ 7.8110 \times 10^{-3} & -6.9744 \times 10^{-3} & 1.0000 \end{bmatrix}$$

(c) Stich the matching key points.

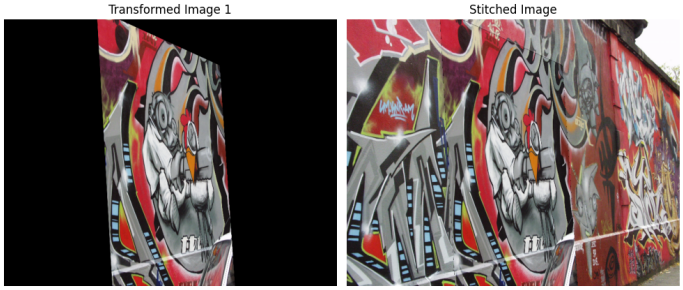

Figure 7: Stiched Plots

5 Github Repository

Link.

Github/EN160_Assignment_02