EN3160 – Image Processing and Machine Vision

Assignment 2 – Fitting and Allignment

Name: Clarance L.G.S.

Index No: 200094R

GitHub Repository: https://github.com/ClaranceLGS/EN3160-Image-Processing-and-Machine-Vision/tree/main/Assignment%202

Question 1

```python
# Define the LoG filter
def LoG(sigma):
    hw = round(3 * sigma) # half width of the filter
    X, Y = np.meshgrid(np.arange(-hw, hw + 1, 1), np.arange(-hw, hw + 1, 1))
    log_filter = 1 / (2 * np.pi * sigma**2) * (X**2 / (sigma**2) + Y**2 / (sigma**2) - 2) * np.exp(-(X**2 +
Y**2) / (2 * sigma**2))
    return log_filter
# Function to find local maxima in the filtered image
def maxima(img_filtered, sigma):
    coordinates = []
    (h, w) = img_filtered.shape
    k = 1
    for i in range(k, h - k):
        for j in range(k, w - k):
            img_patch = img_filtered[i - k:i + k + 1, j - k:j + k + 1]
            if np.max(img_patch) >= 0.09:
                x, y = np.unravel_index(np.argmax(img_patch), img_patch.shape)
                coordinates.append((i + x - k, j + y - k)) # coordinates of the local maxima
    return set(coordinates)
```

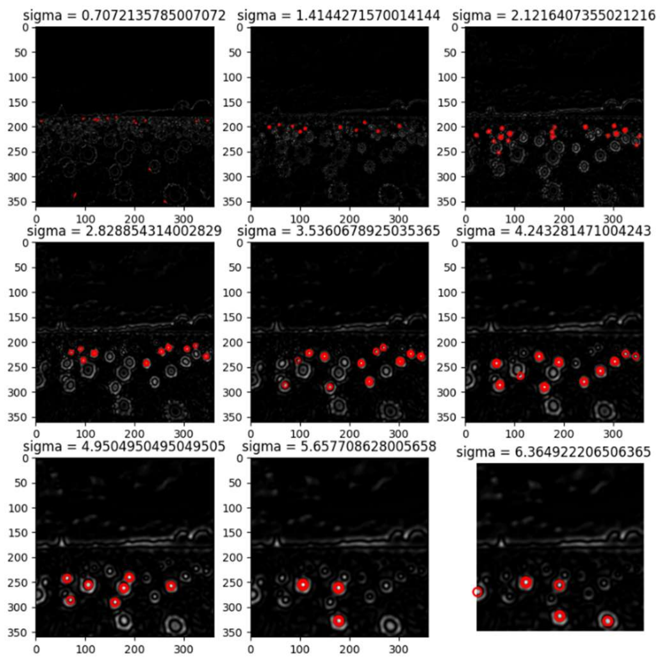

Figure 1: Detected blobs at different sigma values

- In this question we are supposed to detect blobs using Laplacian of Gaussians and scale space extrema detection.
- Range of sigma values – 0.7072 to 6.3649

Question 2

Best line = [0.72354266 0.69027967 1.84109291], No of inliers = 50, best error = 10.425840187164722

Best circle = [1.78614648 3.31319122 10.0848647], No of inliers = 46, best error = 8.667271061567597

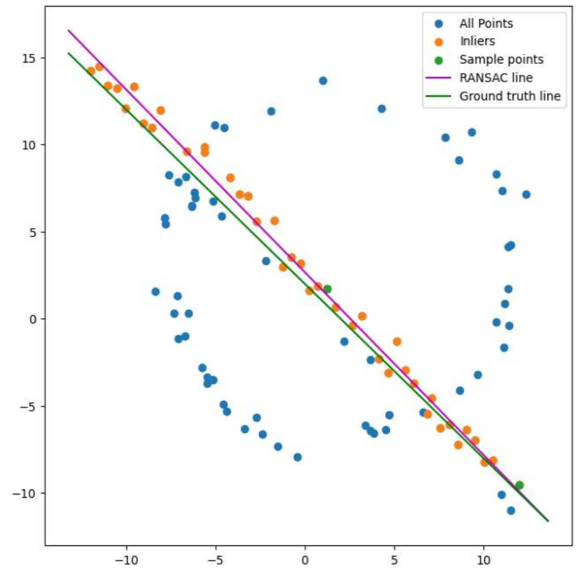

Figure 2: Best line model using RANSAC algorithm.

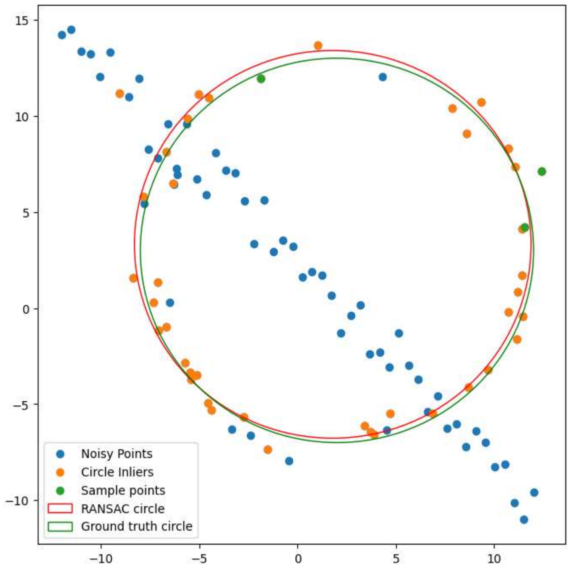

Figure 3: Best Circle model using RANSAC algorithm.

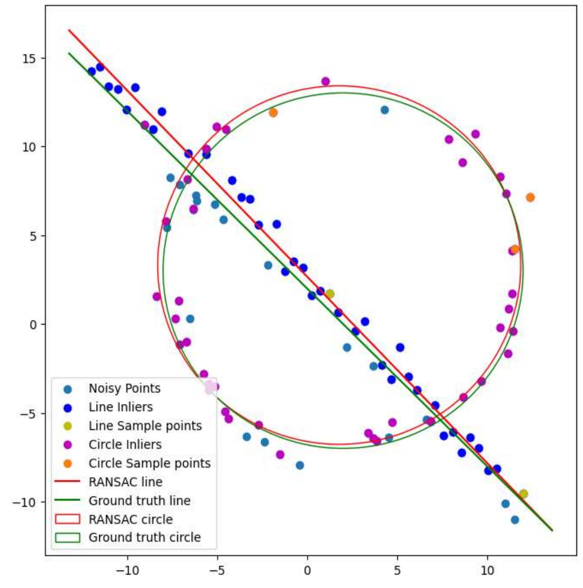

Figure 4: Circle and Line fitting with RANSAC

- In this question we fit a line and a circle to set of noisy points that conform to a line and a circle.

- Ans(d):

Fitting the circle before the line in a RANSAC-based approach can introduce inaccuracies and compromise the quality of both fits. This occurs because RANSAC may inadvertently select random points from the line structure as part of the circle fitting process, leading to suboptimal results for the circle fit. Conversely, fitting the line first and then the circle allows for the removal of line inliers, enhancing data quality for the subsequent circle fit. This sequential approach minimizes the influence of incorrect inliers, resulting in more accurate and robust fits for both geometric models.

Question 3

```
# Mouse Callback function
def click_event(event, x, y, flags, param):
    global points_img_1
    if event == cv.EVENT_LBUTTONDOWN: # If the left mouse button is clicked, record the point
        points_img_1.append((x, y))
        cv.circle(img_1, (x, y), 5, (0, 255, 0), -1)
        cv.imshow("Image 1", img_1)
        if len(points_img_1) == 4: # If four points are selected, proceed with further processing
            compute_homography()
# Function to compute homography matrix and perform superimposition
def compute_homography():
    global points_img_1
    # Define the corresponding points in img_2 with the same aspect ratio as image1)
    points_img_2 = np.array([[0, 0], [img_2.shape[1], 0], [img_2.shape[1], img_2.shape[0]], [0,
img_2.shape[0]]], dtype=np.float32)
    # Compute the homography matrix
    homography_matrix, _ = cv.findHomography(points_img_2, np.array(points_img_1, dtype=np.float32))
    # Warp img_2 to match the perspective of img_1
    warped_img_2 = cv.warpPerspective(img_2, homography_matrix, (img_1.shape[1], img_1.shape[0]))
    # Blend the warped img_2 onto img_1
    alpha = 0.4 # Blending parameter
    blended_img = cv.addWeighted(img_1, 1 - alpha, warped_img_2, alpha, 0)
```

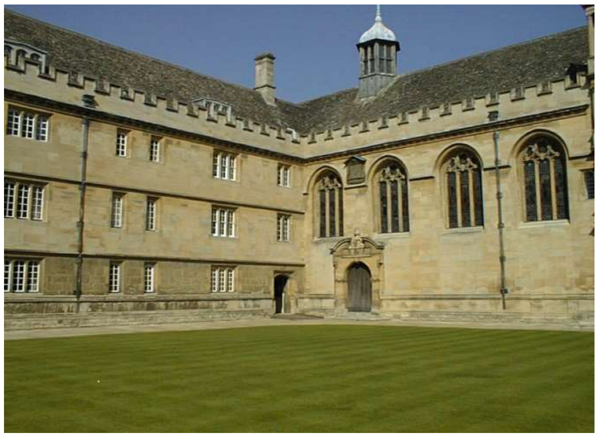
Figure 5: Image 1

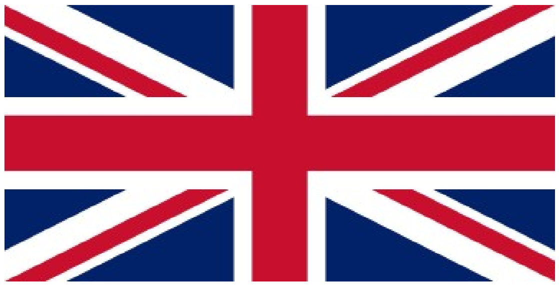
Figure 6: Image 2

- Here we wrapping one image(Image 2) and super imposing it to another image(Image 1).

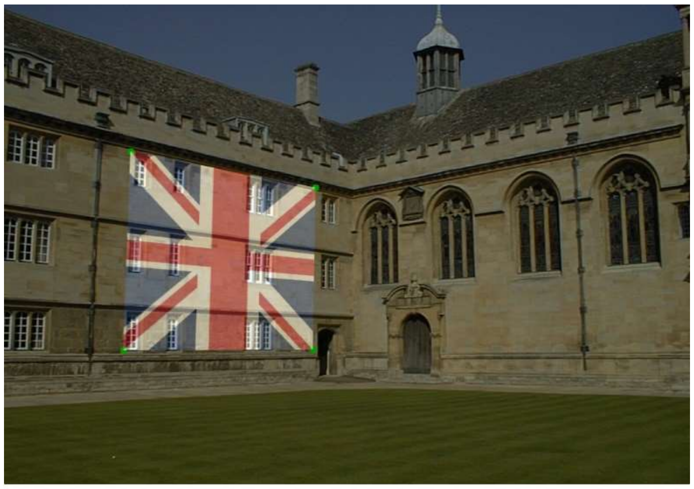
Figure 7: Super Imposed Image

Question 4

- Here we stich Image 1and Image 2 together.
- For this we created keypoints and descriptors for both images and then matched them using a matcher.
- Then we compute homography matrix and warp the image to other image.
- We initiate SOFT for creating keypoints and descriptors.
- We use RANSAC for finding the best homography matrix along with BF matcher for matching the descriptors.

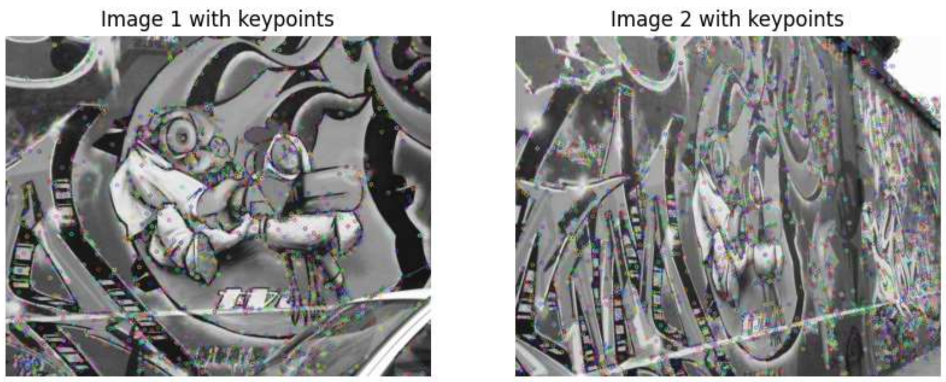

Figure 8: Images with key points

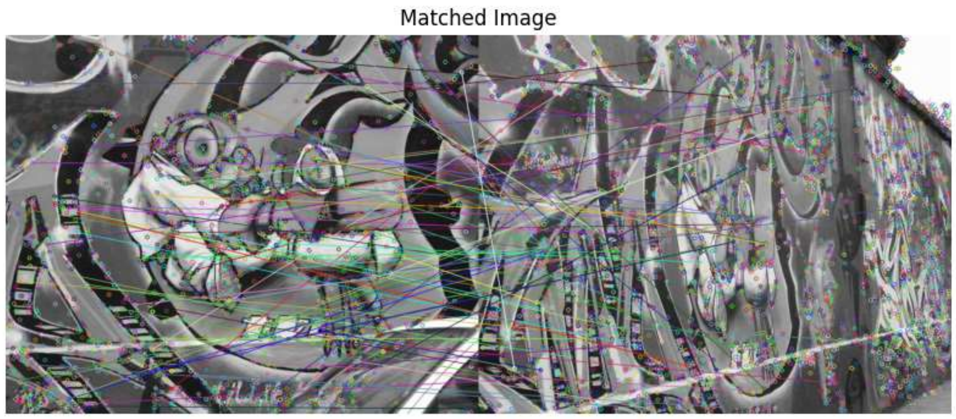

Figure 9:Matched Image

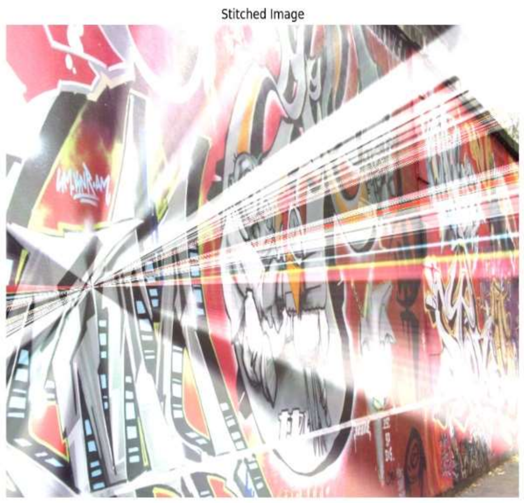

Figure 11: Stiched Image

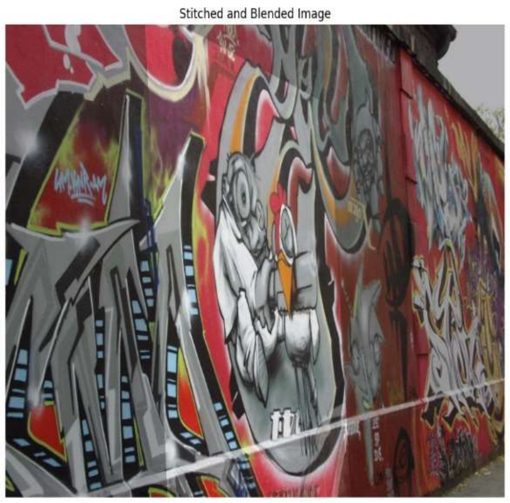

Figure 10: Blended Image