Name – P.B.K. Sanjana Gimhani Kapukotuwa

Index Number – 200287L

EN3160 Assignment 2 on Fitting and Alignment

GitHub repo: https://github.com/sanjanakapukotuwa/EN3160-Image_Processing_and_Machine_Vision

Question 1

Range of sigma used: 1-3

Radius of the largest circle: 25 , corresponding sigma value = 3


```
# Create an empty list to store detected circles
circles = []

# Loop through different sigma values to detect blobs at different scales
for sigma in np.linspace(min_sigma, max_sigma, num_sigma):

    # Print the current sigma value to the console
    print(f"Current sigma value: {sigma}")

    # Apply LoG (Laplacian of Gaussian) to the grayscale image with the current sigma
    blurred = cv2.GaussianBlur(gray_image, (0, 0), sigma)
    laplacian = cv2.Laplacian(blurred, cv2.CV_64F)

    # Calculate the absolute Laplacian values
    abs_laplacian = np.abs(laplacian)

    # Create a binary image where blobs are detected using the threshold
    blob_mask = abs_laplacian > threshold * abs_laplacian.max()

    # Find contours in the blob mask
    contours, _ = cv2.findContours(blob_mask.astype(np.uint8), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Loop through the detected contours and fit circles to them
    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))
```

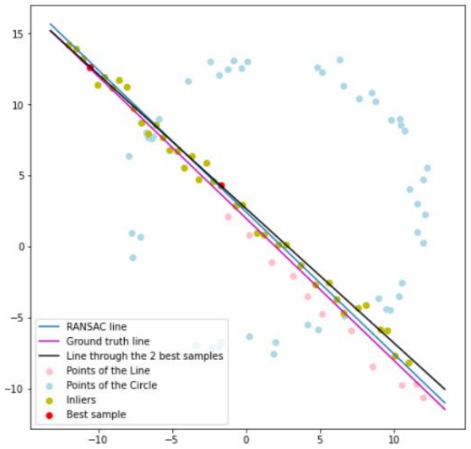
Question 2

The steps of estimating the best model for a line/circle using the RANSAC algorithm are,

- select s number of points at random
- fit the model to those s points
- find the inliers of that model that lie within the given threshold t
- if the number of inliers is greater than the minimum amount required, refit the model considering all the inliers
- repeat until we find the best model

Best fit line is estimated using the inliers corresponding to the best sample using scipy.optimize.minimize

```
while iteration < max_iterations:
    indices = np.random.randint(0, N, s) # A sample of two (s) points selected at random
    x0 = np.array([1, 1, 0]) # Initial estimate
    res = minimize(fun = line_tls, args = indices, x0 = x0, tol= 1e-6, constraints=cons, options={'disp': True})
    inliers_line = consensus_line(X_, res.x, t) # Computing the inliers
    print('rex.x: ', res.x)
    print('Iteration = ', iteration, '. No. inliners = ', inliers_line.sum())

    # Only compute the model if number of inliers > d

    if inliers_line.sum() > d:
        x0 = res.x

        # Computing the new model using the inliers
        res = minimize(fun = line_tls, args = inliers_line, x0 = x0, tol= 1e-6, constraints=cons, options={'disp': True})

        print(res.x, res.fun)
        if res.fun < best_error:
            print('A better model found ... ', res.x, res.fun)
            best_model_line = res.x
            best_eror = res.fun
            best_sample_line = X_[indices,:]
            res_only_with_sample = x0
            best_inliers_line = inliers_line

    iteration += 1
```

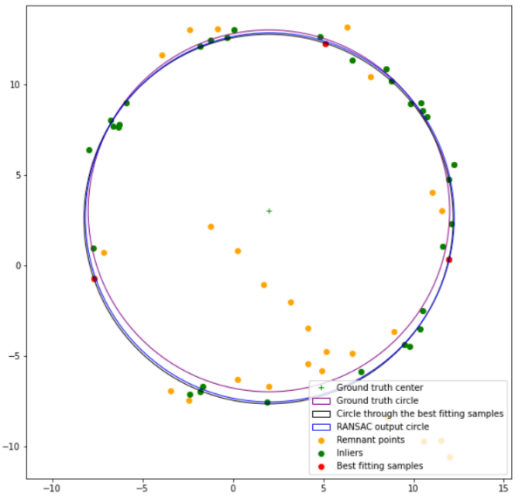


Best fit circle is estimated using the inliers and center point corresponding to the initial circle passing through the best 3 samples using scipy.optimize

```
# Calculting the RANSAC outputs for the data set
# Calculate the center and radius of the
#circle passing through the 3 best samples

center, radius, sample, inliers = RANSAC_Circle(X_new, 10000)
print(center)
```

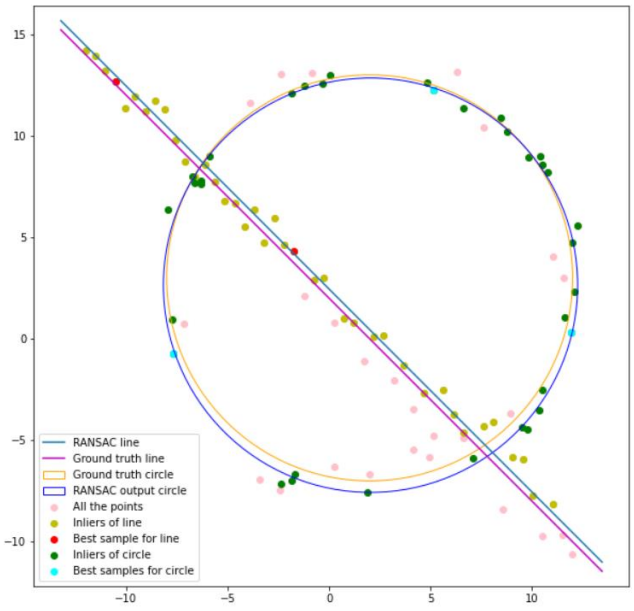
```
# Now lets run a code snippet to substract the consensus of the best fit line

remaining_indices = [i for i in range(len(best_inliers_line)) if best_inliers_line[i]==False]
#print (remaining_indices)
remaining_points_of_line = X_line[remaining_indices]
#print(remaining_points_of_line)

# The remaining points of the line will be added to the points of the circle
X_new = np.vstack((X_circ, remaining_points_of_line))
```

```
# Estimate the circle using the inliers using the least square method
ransac_center, ransac_radius = estimateCircle(center[0], center[1], inliers)

print("Center of RANSAC =", ransac_center)
print("Radius of RANSAC =", ransac_radius)
```



The ground truth line/circle, model passing through the sample points, best fitting model using the inliers of both the line and circle are plotted separately below.

(c) Final models



(d) If we try to fit the circle at first, the model will consider the points corresponding to the line to be points on the radius of the circle as well and estimate a circle with a very large radius passing through those points. This will lead to an incorrect model.

Question 3

The below code computes a homography that maps the flag image to this plane, and warps the flag, and blends it on to the source image.

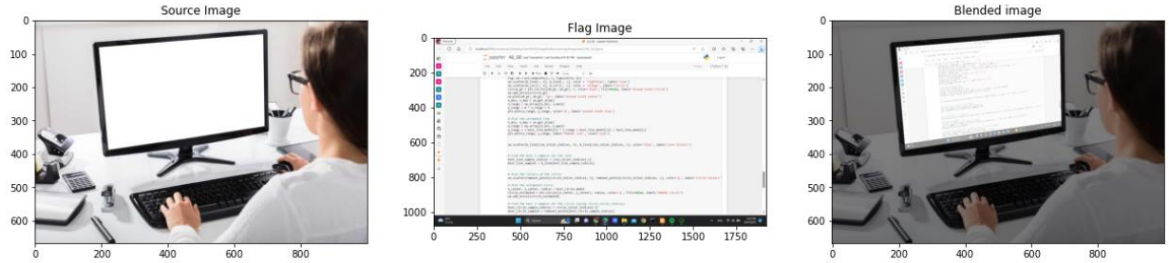
```
points_main_image = np.array([[x1, y1], [x2, y2], [x3, y3], [x4, y4]], dtype=np.float32)

# Define four points of the wallpaper image (assuming it's a rectangle)
#img_width, img_height, _ = wallpaper_image.shape
img_height, img_width, _ = wallpaper_image.shape
points_wallpaper = np.array([[0, 0], [img_width, 0], [img_width, img_height], [0, img_height]], dtype=np.float32)

# Compute the homography matrix
homography_matrix, _ = cv2.findHomography(points_wallpaper, points_main_image)

# Warp the wallpaper image to match the computer's screen
wallpaper_warped = cv2.warpPerspective(wallpaper_image, homography_matrix, (pc_image.shape[1], pc_image.shape[0]))

# Blend the warped wallpaper onto the image of the pc
blended_image = cv2.addWeighted(pc_image, 0.5, wallpaper_warped, 0.5, 0)
```

The below code snippet was used to identify the 4 points in the source image on which the flag image must be superimposed.

```
def click_event(event, x, y, flags, params):

    # checking for left mouse clicks
    if event == cv2.EVENT_LBUTTONDOWN:

        # displaying the coordinates
        # on the Shell
        print(x, ' ', y)

        # displaying the coordinates
        # on the image window
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(img, str(x) + ',' +
                    str(y), (x,y), font,
                    1, (255, 0, 0), 2)
        cv2.imshow('image', img)
```

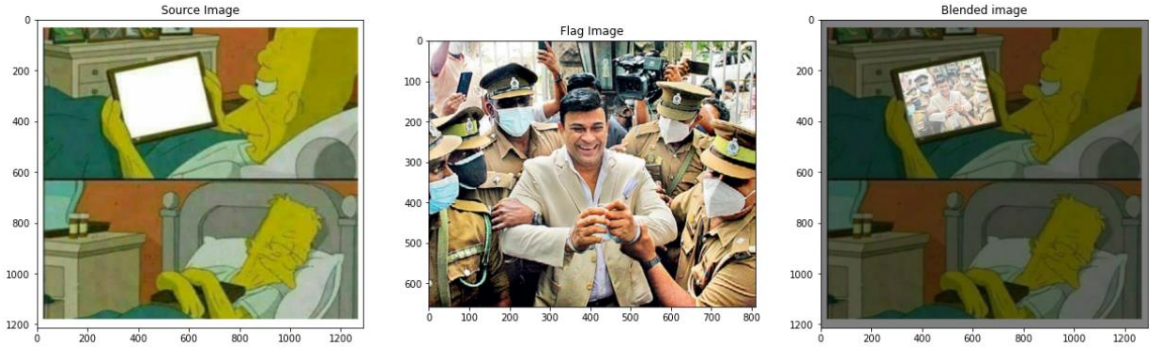
```
# driver function
if __name__=="__main__":

    # reading the image
    img = cv2.imread('blank_picture_frame.jpg', 1)

    # displaying the image
    cv2.imshow('image', img)

    # setting mouse handler for the image
    # and calling the click_event() function
    cv2.setMouseCallback('image', click_event)

    # wait for a key to be pressed to exit
    cv2.waitKey(0)

    # close the window
    cv2.destroyAllWindows()
```

I clicked a screenshot of my laptop screen and superimposed with the source image so that it seems that the woman in working on some code on her computer.



A childhood image of me superimposed with an image of an empty picture frame kept on a table.

At last but not least, why not use this method to generate some meme material. 😉



Question 4

Few functions were defined to compute and match sift features and to compute the homography.

The below code snippet was used to call these functions and to stitch the two graffiti images. Output is also displayed below.

```
match1_2 = custom_sift_match(img1,img2)
match2_3 = custom_sift_match(img2,img3)
match3_4 = custom_sift_match(img3,img4)
match4_5 = custom_sift_match(img4,img5)
match1_5 = custom_sift_match(img1,img5)

# Generate Correspondence Matrices
correspondenceMatrix1_2 = np.matrix(match1_2)
correspondenceMatrix2_3 = np.matrix(match2_3)
correspondenceMatrix3_4 = np.matrix(match3_4)
correspondenceMatrix4_5 = np.matrix(match4_5)

# Run ransac algorithm
H1_2 = RANSAC(correspondenceMatrix1_2)
H2_3 = RANSAC(correspondenceMatrix2_3)
H3_4 = RANSAC(correspondenceMatrix3_4)
H4_5 = RANSAC(correspondenceMatrix4_5)

# Reshaping the homography to the matric format
H1_2 = custom_homography_reshape(H1_2)
H2_3 = custom_homography_reshape(H2_3)
H3_4 = custom_homography_reshape(H3_4)
H4_5 = custom_homography_reshape(H4_5)

H1_5 = H4_5@H3_4@H2_3@H1_2
H1_5 = custom_homography_reshape(H1_5)
print(H1_5)

plot_warped(img1,img5,H1_5)
```

Homography obtained from the calculation.

```
[[ 6.70443673e+00 -7.22470754e+00 -1.39447473e+00]
 [ 5.29689825e+00 -5.17070305e+00 -2.37703088e+02]
 [ 1.02250666e-02 -1.32862858e-02  1.00000000e+00]]
```