Assignment 2 - Fitting and Alignment

EN3160: Image Processing And Machine Vision

Index No. - 210549D

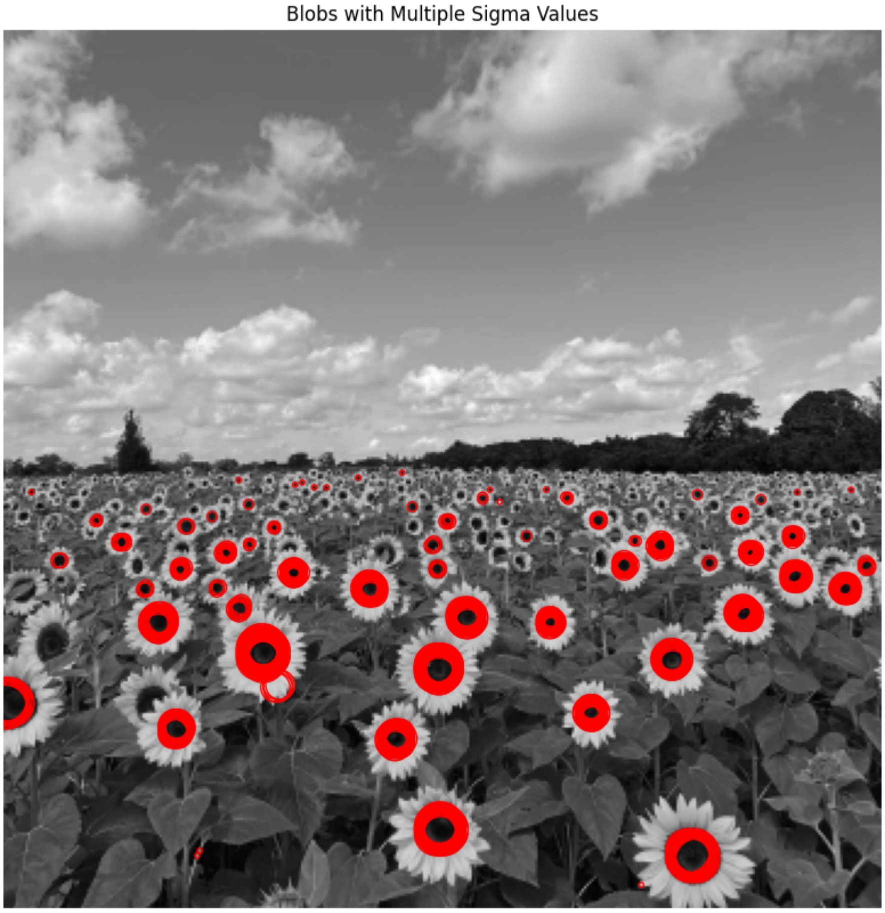
Name - N.P.S.S. Rupasinghe

GitHub Link for Code

Question 1: Blob Detection

- Range of values: 1 to 10

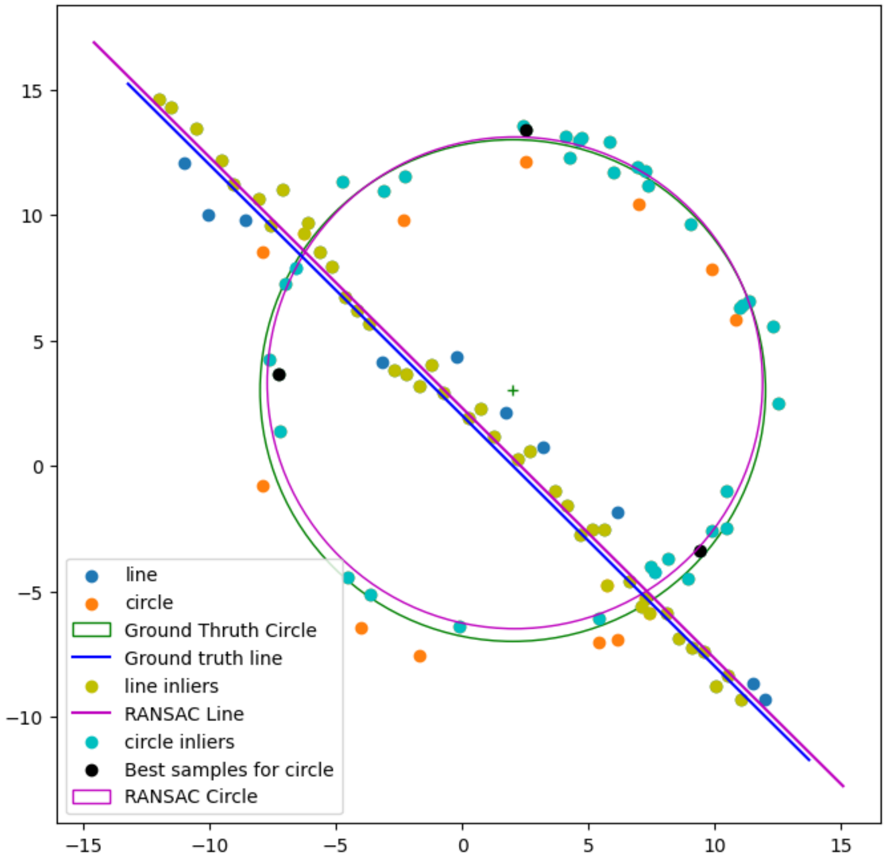
```
def laplacian_of_gaussian_kernel(sigma):
    size = int(np.ceil(3 * sigma) * 2 + 1)
    # Create a 2D grid of (x, y) coordinates
    ax = np.linspace(-(size // 2), size // 2, size)
    xx, yy = np.meshgrid(ax, ax)
    # Calculate the LoG using the formula
    squared_distances = xx**2 + yy**2
    log = (-1/(np.pi * sigma**4)) * (1 - (squared_distances / (2 * sigma**2)))
    * np.exp(-squared_distances / (2 * sigma**2))
    return log

def apply_log_filter_cv(image, sigma):
    log_kernel = laplacian_of_gaussian_kernel(sigma)
    # Apply the LoG filter using OpenCV's filter2D
    # -1 indicates the depth of the output image is the same as the source
    filtered_image = cv.filter2D(image, ddepth=-1, kernel=log_kernel)
    return filtered_image
```



Blobs with Multiple Sigma Values

Question 2: RANSAC

Plot



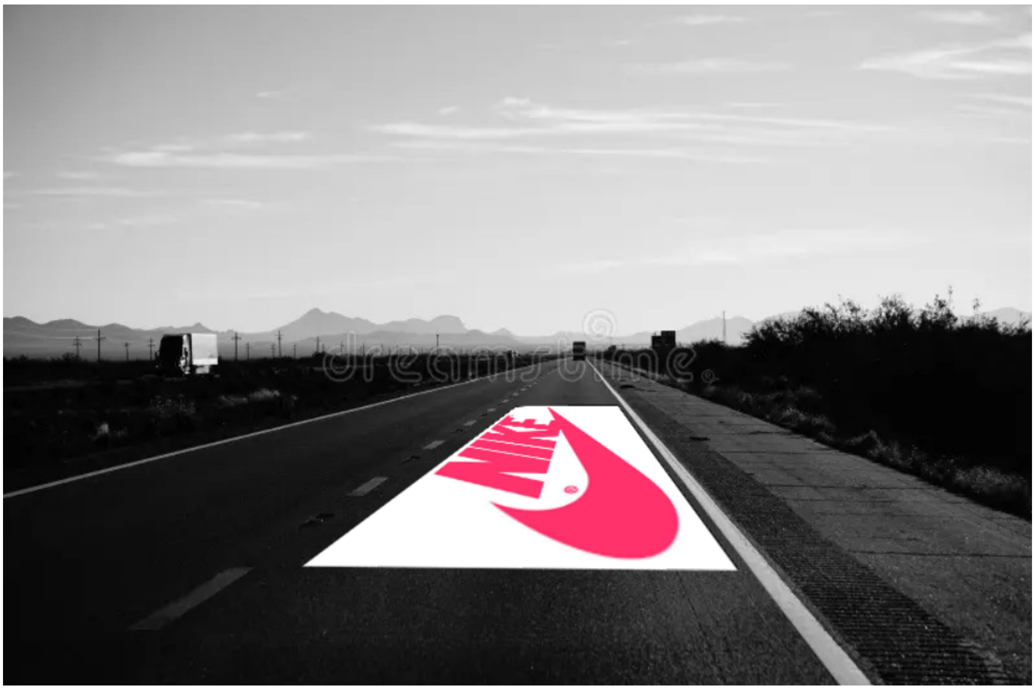
What will happen if we fit the circle first?

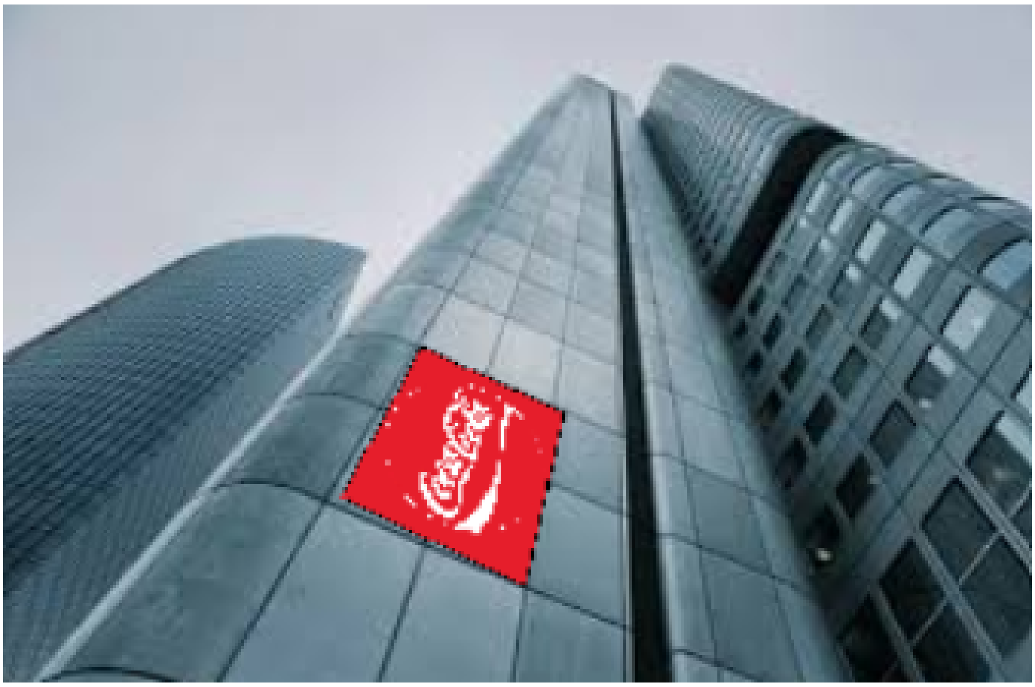
If the circle is fitted first, there is a chance of the three random sample points all being on the line. Then the calculated circle will be large and locally similar to a line. However, since the RANSAC algorithm will run for many iterations with different sample points, it is still possible to accurately fit the circle without removing the line points

Line Code

```
def ransac_line(X, line_max_iterations=100, line_threshold=1.0, line_data_fraction=0.4, line_sample=2):
    N = X.shape[0]  # Number of points
    line_data_points = int(line_data_fraction * N)  # Minimum number of inliers for consensus
    line_best_error = np.inf  # Track the best error
    best_line_model = []  # Initialize the best line model
    best_line_inliers = []  # Initialize the best inliers
    def line_total_error(x, line_indices):  # Total least squares error function
        a, b, d = x[0], x[1], x[2]
        return np.sum(np.square(a * X[line_indices, 0] + b * X[line_indices, 1] - d))
    def cons(x):  # Constraint: ensure [a, b]^T is a unit vector
        return x[0] ** 2 + x[1] ** 2 - 1
    constraint = ({'type': 'eq', 'fun': cons})  # Define the constraint
    def line_consensus_set(X, x, line_threshold):  # Find points within threshold distance to the line
        a, b, d = x[0], x[1], x[2]
        error = np.abs(a * X[:, 0] + b * X[:, 1] - d)
        return error < line_threshold
    for line_iteration in range(line_max_iterations):  # RANSAC iterations
        line_indices = np.random.randint(0, N, line_sample)  # Select random point samples
        x0 = np.array([1, 1, 0])  # Initial guess for [a, b, d]
        res = minimize(line_total_error, x0=x0, args=(line_indices,), tol=1e-6, constraints=constraint)  # Minimiz
        line_inliers = line_consensus_set(X, res.x, line_threshold)  # Find the inliers
        if np.sum(line_inliers) > line_data_points:  # Check if inliers exceed threshold
            x0 = res.x  # Update initial guess with the current model
            res = minimize(line_total_error, x0=x0, args=(line_inliers,), tol=1e-6, constraints=constraint)  # Ref
            if res.fun < line_best_error:  # Update best model if error improves
                line_best_error = res.fun
                best_line_model = res.x
                best_line_inliers = line_inliers
    return best_line_model, best_line_inliers  # Return the best model and inliers
```

Question 3: Superimposing

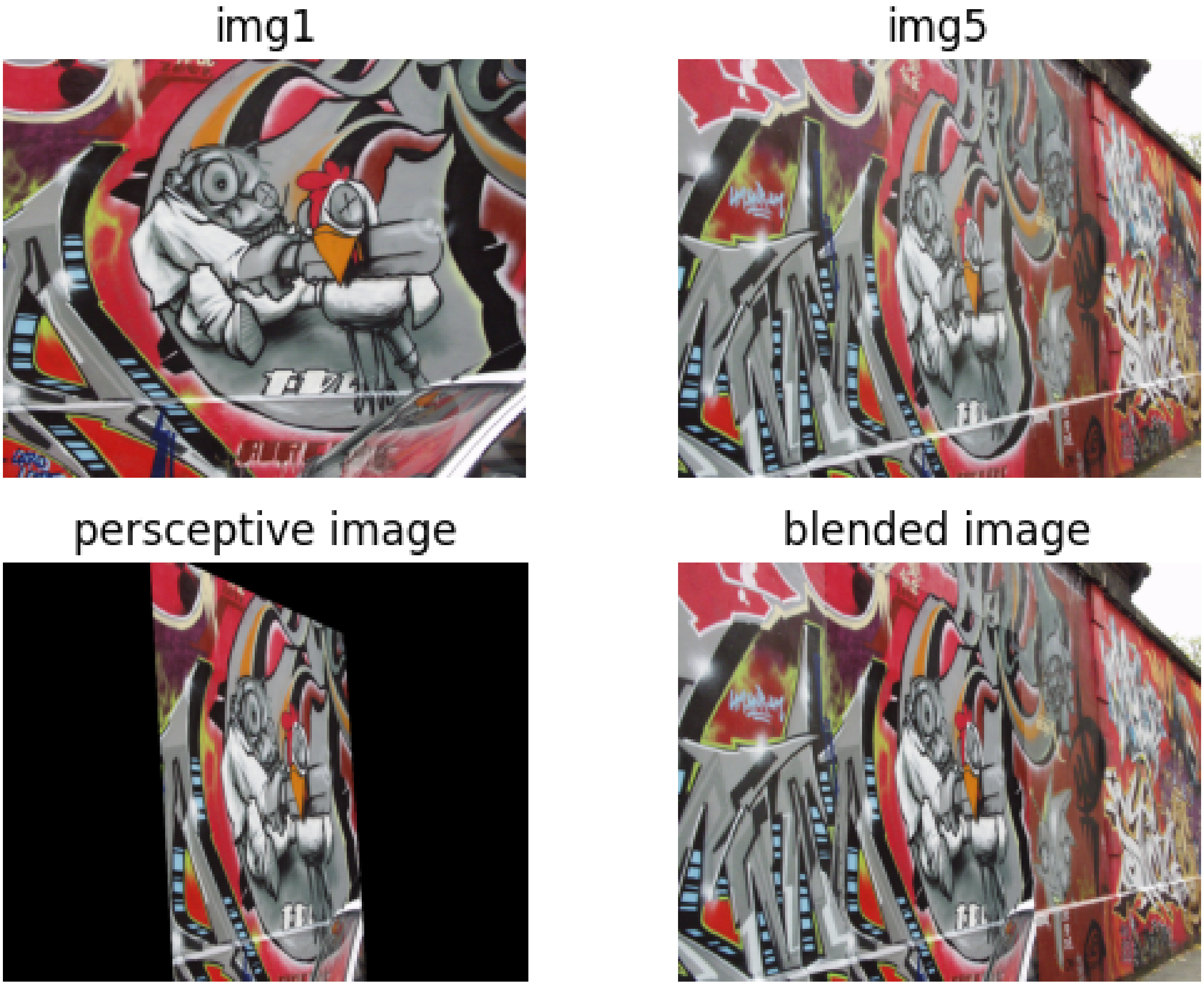




Question 4: Image Stitch

Stitch img1.ppm onto img5.ppm

There weren't enough matching features between images 1 and 5 for a reliable homography. Instead, homographies were computed between consecutive pairs (1-2, 2-3, etc.), leveraging their higher similarity. Image 1 was then gradually transformed through these homographies to align with the coordinate space of image 5.



img1

img5

persceptive image

blended image