EN3160 – Image Processing and Machine Vision

Name:	Index No.:	Submission Date:
P.N. Kulasingham	210303U	23.10.2024

GitHub Repository: Assignment 02 link

Question 1

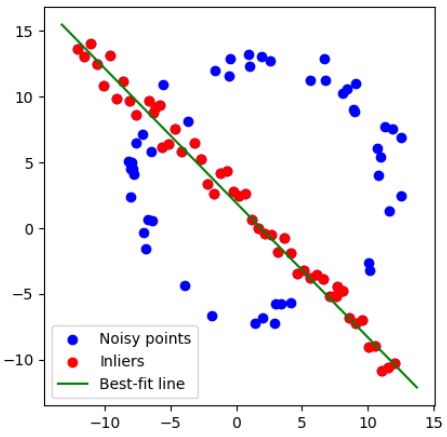
```
def detect_blobs(image):
    img1_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY) / 255
    fig, axes = plt.subplots(1, 2, figsize = (10, 10))
    axes[0].imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
    axes[0].set_title("Original Image")
    axes[0].axis('off')
    ax = axes[1]
    ax.imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
    patches = []; labels = []; circles = []
    for r in range(1,11):
        sigma = r/1.414
        LOG = sigma**2 * laplacian_of_gaussian(sigma)
        img1_log = np.square(cv.filter2D(img1_gray, -1, LOG))
        coordinates = detect_max(img1_log, sigma)
        for x, y in coordinates:
            radius = sigma * 1.414
            circles.append((x, y, radius))
            c = plt.Circle((y, x), sigma * 1.414, color='blue', linewidth=0.5, fill=False)
            ax.add_patch(c)
        patches.append(c)
        labels.append(f'r = {r}')
        ax.plot()
    ax.set_xlim(0, image.shape[1]); plt.axis('off'); ax.set_title("Image with blobs")
    plt.show()
    if circles: largest_circle = max(circles, key=lambda c: c[2])  # Find the largest
        circle by radius
    print(f"Largest circle detected: Center=({largest_circle[0]}, {largest_circle[1]}),
        Radius={largest_circle[2]}")
    return
```

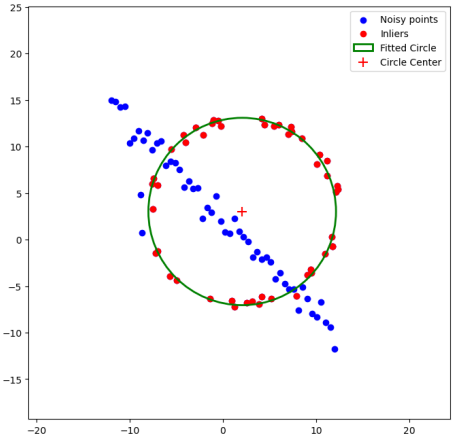

```
Largest circle detected: Center=(341, 280), Radius=10.0
```

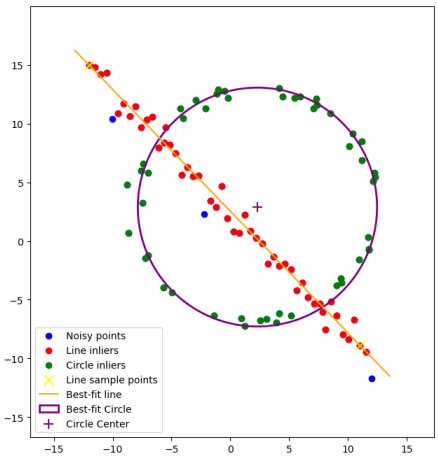
Question 2

```python
def ransac_line(X, threshold, max_iter=1000):
    best_inliers = []
    best_model = None

    for _ in range(max_iter):
        sample = X[np.random.choice(X.shape[0], 2, replace=False)]
        p1, p2 = sample
        line = np.cross(np.append(p1, 1), np.append(p2, 1))
        a, b, d = line[0], line[1], line[2]
        norm_factor = np.sqrt(a**2 + b**2)
        a, b, d = a / norm_factor, b / norm_factor, d / norm_factor
        normal = np.array([a, b])
        distances = np.abs(np.dot(X, normal) + d) / np.linalg.norm(normal)
        inliers = X[distances < threshold]
        if len(inliers) > len(best_inliers):
            best_inliers = inliers
            best_model = (a, b, d)

    return best_model, best_inliers
```

```python
def fit_circle(points):
    def calc_R(xc, yc):
        return np.sqrt((points[:, 0] - xc) ** 2 + (points[:, 1] - yc) ** 2)
    def f_2(c):
        Ri = calc_R(*c)
        return np.std(Ri)
    center_estimate = np.mean(points, axis=0)
    center_optimized = minimize(f_2, center_estimate).x
    radii = calc_R(*center_optimized)
    return center_optimized[0], center_optimized[1], np.mean(radii)

def find_inliers(points, xc, yc, radius, threshold):
    distances = np.sqrt((points[:, 0] - xc) ** 2 + (points[:, 1] - yc) ** 2)
    inliers = points[np.abs(distances - radius) < threshold]
    return inliers
```



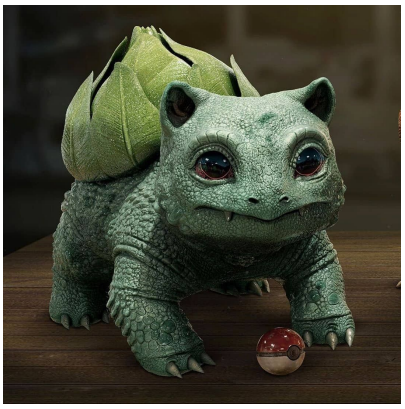


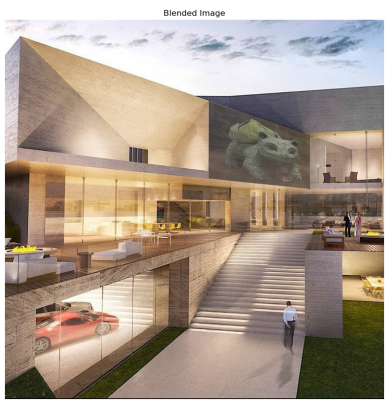


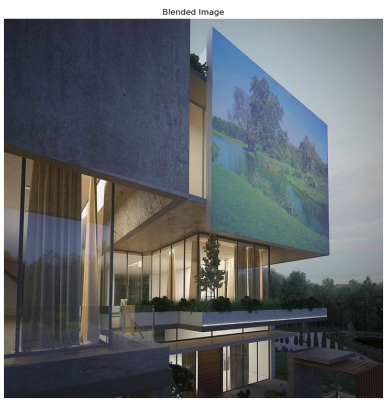
If we fit the circle first, we end up selecting points from both the actual circle and the line in our initial sample. Since RANSAC looks for consensus (inliers) based on the model being fitted, this can lead to a poor circle estimate because points from the line will interfere with the circle fitting process.

Question 3

```python
def blend_image(img1, img2,points):
    flag_height, flag_width = img2.shape[:2]
    flag_corners = np.array([[0, 0], [flag_width, 0], [flag_width, flag_height],
        [0, flag_height]], dtype="float32")
    points = np.array(points, dtype="float32")
    homography_matrix, _ = cv.findHomography(flag_corners, points)
    warped_img2 = cv.warpPerspective(img2, homography_matrix, (img1.shape[1],
        img1.shape[0]))
    if warped_img2.shape[2] == 4:
        flag_mask = (warped_img2[..., 3] > 0).astype(np.uint8)
        warped_img2_rgb = warped_img2[..., :3]
    else:
        flag_mask = np.ones_like(warped_img2[..., 0], dtype=np.uint8)
        warped_img2_rgb = warped_img2
    transparency_factor = 0.3
    adjusted_flag_mask = flag_mask * transparency_factor
    adjusted_flag_mask = np.clip(adjusted_flag_mask, 0, 1)

    # Blend the images with the adjusted transparency
    blended_image = img1.copy()
    for c in range(0, 3):
        blended_image[..., c] = (
            blended_image[..., c] * (1 - adjusted_flag_mask) +
            warped_img2_rgb[..., c] * adjusted_flag_mask
        )
    return blended_image
```







Question 4


```
Homography Matrix
[[ 6.18057912e-01  6.07853414e-02  2.19382479e+02]
 [ 2.16738670e-01  1.15694927e+00 -2.52514934e+01]
 [ 4.83851717e-04 -4.03656168e-05  9.93068043e-01]]

Given Homography Matrix
[[ 6.2544644e-01  5.7759174e-02  2.2201217e+02]
 [ 2.2240536e-01  1.1652147e+00 -2.5605611e+01]
 [ 4.9212545e-04 -3.6542424e-05  1.0000000e+00]]
```

```python
def find_homography(good_matches, keypoints1, keypoints5):
    src_full = []
    dst_full = []
    for match in good_matches:
        src_full.append(np.array(keypoints1[match.queryIdx].pt))
        dst_full.append(np.array(keypoints5[match.trainIdx].pt))
    src_full = np.array(src_full)
    dst_full = np.array(dst_full)
    num_points = 4
    thres = 1
    d = 0.5 * len(good_matches)
    iters = 200
    best_homography = None
    best_inlier_count = 0
    best_inliers = None
    for i in range(iters):
        chosen_matches = np.random.choice(good_matches, num_points, replace = False)
        src_points = []
        dst_points = []
        for match in chosen_matches:
            src_points.append(np.array(keypoints1[match.queryIdx].pt))
            dst_points.append(np.array(keypoints5[match.trainIdx].pt))
        src_points = np.array(src_points)
        dst_points = np.array(dst_points)
        tform = transform.estimate_transform('projective', src_points, dst_points)
        inliers = get_inliers(src_full, dst_full, tform, thres)
        if len(inliers) > best_inlier_count:
            best_inlier_count = len(inliers)
            best_homography = tform
            best_inliers = inliers
    return best_homography, best_inliers
```