Assignment 2 - EN2550

190456K
https://github.com/Avishka-Perera/UoM-S4-EN2550-Assignments

Question 1 - RANSAC to find circle

RANSAC code implementation

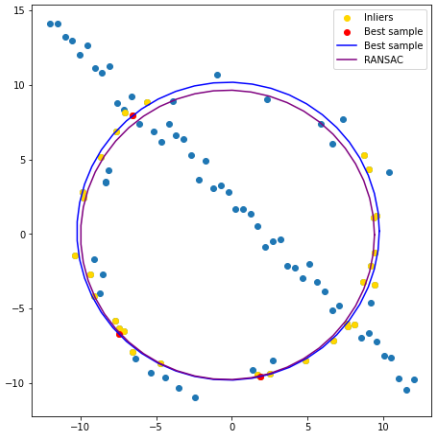
```
e, p, s, d, t = 0.4, 0.99, 3, 60, 0.48
N = np.log(1-p)/np.log(1-(1-e)**s)*10
potential_circles = []
best_fit_index = 0
max_inlier_count = 0
for _ in range(int(N)):
    sample_indices = np.random.randint(0, X.shape[0], 3)
    samples = X[sample_indices]
    r, h, k = findCircle(samples)
    inliers = findInliers(X, r, h, k, t)
    inlier_count = inliers.size
    if inlier_count > d:
        potential_circles.append({"circle": (r,h,k), "inlier_count": inlier_count, "samples": samples,
"inliers": inliers})
        if inlier_count > max_inlier_count:
            max_inlier_count = inlier_count
            best_fit_index = potential_circles.__len__()-1
```

Two main functions for this were used.

1. `findCircle()` returns the radius and the center of the circle when three points are given. Basic coordinate geometry has been used for this.
2. `findInliers()` returns the inlier points in a set of points (`X`) for a given circle (`r, h, k`) and to a given threshold `t`

First, the loop was run N times (calculated by the `np.log(1-p)/np.log(1-(1-e)**s)*10` formula). After that, the best circle with the maximum number of inliers was selected and plotted.

Results



Question 2 - Image warping and blending

Code implementation

```
homography = cv.getPerspectiveTransform(source_points, destination_points)
warped_img = cv.warpPerspective(foreground, homography, background.shape[:2][::-1])
```

```
output = background.copy()
boolean_mat = warped_img != 0
output[boolean_mat] = warped_img[boolean_mat]*opacity + background[boolean_mat]*(1-opacity)
```

- Inbuilt OpenCV functions were used to calculate and apply the homography. The `source_points` were selected as the four corners of the foreground image. `destination_points` were selected by manually inspecting the image by opening it in Windows Paint.
- The warped image was then overlayed by simply excluding all the 0-pixel values (not an ideal way) and placing the rest on top of the background using a boolean matrix according to the weight of the `opacity` ($0 \leq opacity \leq 1$)

Results

Question 3 - Homography Calculation and Image stiching

Good matches were first found using a simple algorithm.

```
def getGoodMatches(im1, im2, checks, match_margin_frac, display=False):

    sift = cv.SIFT_create() # sift object
    index_params = {"algorithm":1, "trees":5} # matcher
    search_params = {"checks":checks}
    flann = cv.FlannBasedMatcher(index_params, search_params)
    kp1, desc1 = sift.detectAndCompute(im1, None)
    kp2, desc2 = sift.detectAndCompute(im2, None)
    matches = flann.knnMatch(desc1, desc2, k=2)
    good_matches = []
    for i, (m,n) in enumerate(matches):
        if m.distance < match_margin_frac*n.distance:
            good_matches.append(m)
    good_matches = np.array(sorted(good_matches, key=lambda x: x.distance))
    return good_matches, kp1, kp2
```

Then RANSAC was used to filter out the right matches

```
def applyRANSAC(matches, kp1, kp2, errorDistFrac):
    N = matches.size
    queryPoints, trainPoints = getAllPoints(matches, kp1, kp2)
    bestHomography = np.zeros((3,3))
    maxInlierCount = 0
    for _ in range(N):
        homography = getRandomHomography(matches, kp1, kp2)
        transformedPoints = applyHomography(homography, trainPoints)
        errorDist = getErrorDist(im1, errorDistFrac)
        inlierCount = getInlierCount(queryPoints, transformedPoints, errorDist)
        if inlierCount > maxInlierCount:
            maxInlierCount = inlierCount
            bestHomography = homography
    return bestHomography, maxInlierCount
```

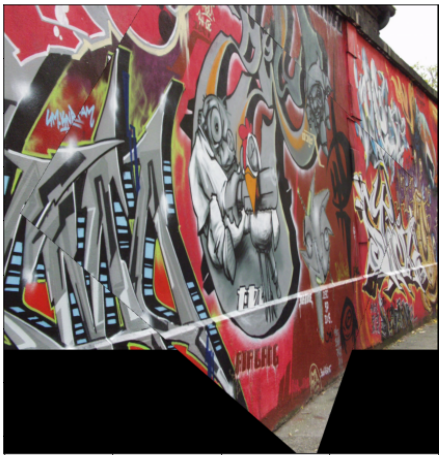
The following code was used to stitch the warped image onto the background. A boolean matrix was used again.

```
def overlayTransformed(background, foreground, homography):
    stiched = np.zeros(foreground.shape).astype(background.dtype)
    stiched[:background.shape[0], :background.shape[1]] = background
    boolean_mat = np.ones(im1.shape)*255
    boolean_mat = cv.warpPerspective(boolean_mat, homography,
(np.array(boolean_mat.shape[:2][::-1])*1.3).astype(int))
    boolean_mat = boolean_mat != 0
    stiched[boolean_mat] = foreground[boolean_mat]
    return stiched
```

This process was applied to the images in the following order as there was a very less number of matches when directly matching `img1.ppm` onto `img5.ppm`

img1.ppm **over** *img3.ppm*

img3.ppm **over** *img4.ppm*

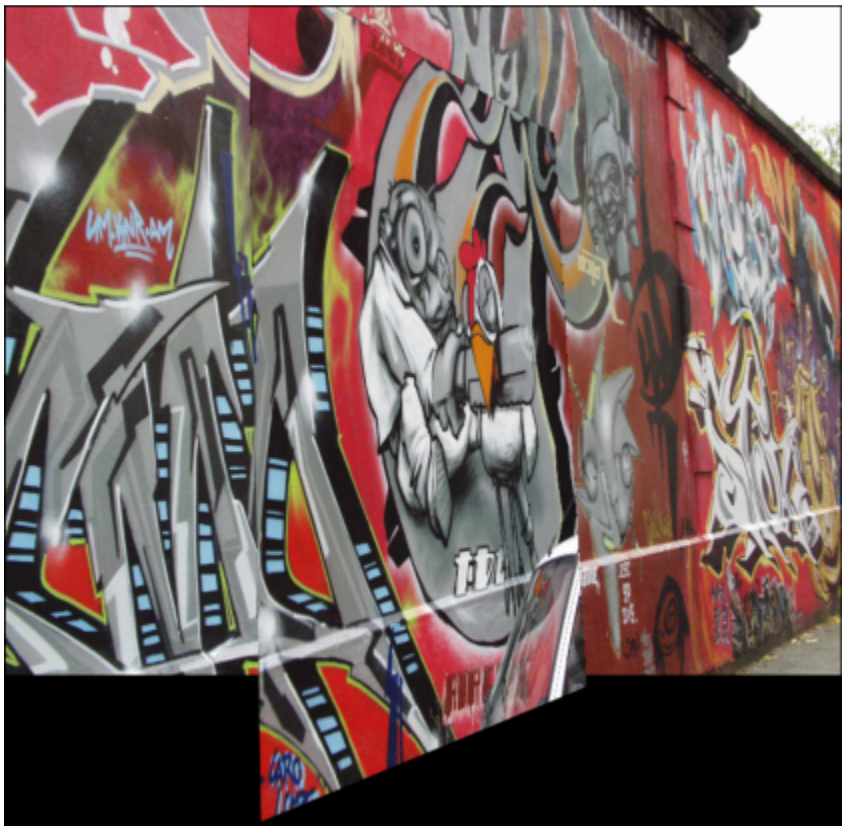


img4.ppm **over** *img5.ppm*

Then the homography to transform `img1.ppm` to `img5.ppm` was calculated as follows

```
homo_1_5 = homo_4_5 @ homo_3_4 @ homo_1_3
```

The final result was as follows:



img1.ppm **over** *img5.ppm*