Exercise 7: Robust model fitting

02504 Computer vision

Morten R. Hannemose, mohan@dtu.dk, DTU Compute

March 15, 2024

Learning objectives

These exercises will introduce you to robust model fitting. You will find straight lines with the Hough transform, and then implement RANSAC to do the same.

Hough Transform

Here you should extract lines from the image `Box3.bmp` from last week, via the Hough transform. The image should be found together with this exercise. Perform the following steps:

Exercise 7.1

Load the image and detect edges in it. Here the function `cv2.Canny` can be used. Visualize the edges you have detected.

Exercise 7.2

Compute the Hough space from the detected edges. Use the function
`hspace, angles, dists = skimage.transform.hough_line(edges)`
What do the returned values `hspace`, `angles`, `dists` mean?

Exercise 7.3

Visualize the Hough space. To get the correct units on the axes you can use

```
extent = [angles[0], angles[-1], dists[-1], dists[0]]
plt.imshow(hspace, extent=extent, aspect='auto')
```

Exercise 7.4

Find peaks in your Hough space, using `skimage.transform.hough_line_peaks`.
`extH, extAngles, extDists = hough_line_peaks(hspace, angles, dists, num_peaks=n)`
Display your identified peaks on top of the Hough space.

Exercise 7.5

Draw the lines that correspond to the identified peaks on top of your original image. For this you can use the `DrawLine` function from week 3 (repeated below)

```
def DrawLine(l, shape):
    #Checks where the line intersects the four sides of the image
    # and finds the two intersections that are within the frame
    def in_frame(l_im):
        q = np.cross(l.flatten(), l_im)
        q = q[:2]/q[2]
        if all(q>=0) and all(q+1<=shape[1::-1]):
            return q
    lines = [[1, 0, 0], [0, 1, 0], [1, 0, 1-shape[1]], [0, 1, 1-shape[0]]]
    P = [in_frame(l_im) for l_im in lines if in_frame(l_im) is not None]
    if (len(P)==0):
        print("Line is completely outside image")
    plt.plot(*np.array(P).T)
```

RANSAC

Here you should fit a line to a data set consisting of inliers with noise and outliers. Such data is generated by the following function:

```
def test_points(n_in, n_out):
    a = (np.random.rand(n_in)-.5)*10
    b = np.vstack((a, a*.5+np.random.randn(n_in)*.25))
    points = np.hstack((b, 2*np.random.randn(2, n_out)))
    return np.random.permutation(points.T).T
```

It is recommended that you do so in the following steps.

Exercise 7.6

Make a function that fits a line, in homogeneous coordinates, given two points.

Exercise 7.7

Make a function that determines which of a set of 2D points are an inliers or outliers with respect to a given line. The threshold should be supplied as parameter to this function, such that it can easily be tuned later

Exercise 7.8

Make a function that calculates the consensus, i.e. the number of inliers, for a line with respect to a set of points

Exercise 7.9

Make a function that randomly samples two of n 2D points (without replacement).

Exercise 7.10

Assemble the functions made above to a working RANSAC algorithm for estimating lines. Set the number of iterations and the threshold manually.

Exercise 7.11

Experiment with the algorithm, what is a good threshold for distinguishing between inliers and outliers?

Exercise 7.12

Add the final step to your implementation, where you fit a new line to all inliers of the best line. The total least squares fit of a straight line to a set of points is given by the first principal component of them. Consider using the code below to get a homogeneous line along the first principal component.

```
def pca_line(x): # assumes x is a (2 x n) array of points
    d = np.cov(x)[:, 0]
    d /= np.linalg.norm(d)
    l = [d[1], -d[0]]
    l.append(-(l@x.mean(1)))
    return l
```

Exercise 7.13

Implement the stopping criteria for RANSAC as described on the slides. Use $p = 0.99$.