CS-E4850 Computer Vision
Exercise Round 5

Alex Herrero Pons

Spring Term Course 2020-2021

Exercise 1. Total least squares line fitting. (Pen and paper problem)
An overview of least squares line fitting is presented on the slide 13 of Lecture 4. Study it in detail and present the derivation with the following stages:

1) Given a line $ax+by-d=0$, where the coefficients are normalized so that $a^2+b^2=1$, show that the distance between a point (x_i, y_i) and the line is $|ax_i + by_i - d|$.

Solution. Let $p_0 = (x_i, y_i)$. Also let $p_1 = (x_i, y_i + d_1)$ and $p_2 = (x_i + d_2, y_i)$ be two points of the given line where d_1 and d_2 are the distances from p_0 to p_1 and p_2 respectively. We have that

$$a(x_i + d_2) + by_i - d = 0$$

$$ax_i + b(y_i + d_1) - d = 0$$

$$-ad_2 = -bd_1 = ax_i + by_i - d$$

$$|ad_2| = |bd_1| = |ax_i + by_i - d|$$

$$d_1 = \frac{a}{b}d_2$$

Now consider the area of the triangle formed by the three given points p_0, p_1 and p_2,

$$A = \frac{1}{2}|d_1 d_2| = \frac{1}{2}\sqrt{d_1^2 + d_2^2}d_0$$

where d_0 is the minimum distance (perpendicular) from the point p_0 to the line and $\sqrt{d_1^2 + d_2^2}$ is the distance between the points p_1 and p_2.
So we have that

$$|d_1 d_2| = \sqrt{d_1^2 + d_2^2}d_0$$

$$d_0 = \frac{|d_1 d_2|}{\sqrt{d_1^2 + d_2^2}}$$

And since $d_1 = \frac{a}{b}d_2$ we obtain that

$$d_0 = \frac{|\frac{a}{b}d_2 d_2|}{\sqrt{(\frac{a}{b}d_2)^2 + d_2^2}} = \frac{|\frac{a}{b}d_2^2|}{\sqrt{\frac{a^2}{b^2}d_2^2 + d_2^2}} = \frac{|\frac{a}{b}d_2^2|}{\sqrt{\frac{d_2^2}{b^2}(a^2 + b^2)}} =$$

$$= \frac{|\frac{a}{b}d_2^2|}{\frac{d_2}{b}\sqrt{a^2+b^2}} = \frac{|ad_2|}{\sqrt{a^2+b^2}} = \frac{|ax_i+by_i-d|}{\sqrt{a^2+b^2}}$$

So, knowing that $a^2+b^2=1$ we obtain that

$$d_0 = |ax_i+by_i-d|$$

2) Thus, given n points (x_i, y_i), $i = 1, \ldots, n$, the sum of squared distances between the points and the line is $E = \sum_{i=1}^{n}(ax_i + by_i - d)^2$. In order to find the minimum of E, compute the partial derivative $\partial E/\partial d$, set it to zero, and solved in terms of a and b.

Solution.

$$E = \sum_{i=1}^{n}(ax_i+by_i-d)^2$$

$$\frac{\partial E}{\partial d} = \sum_{i=1}^{n} -2(ax_i+by_i-d) = 0$$

$$-2\sum_{i=1}^{n}(ax_i+by_i) = -2\sum_{i=1}^{n} d$$

$$a\sum_{i=1}^{n} x_i + b\sum_{i=1}^{n} y_i = dn$$

$$d = \frac{a}{n}\sum_{i=1}^{n} x_i + \frac{b}{n}\sum_{i=1}^{n} y_i = a\overline{x} + b\overline{y}$$

3) Substitute the expression obtained for d to the formula of E, and show that then $E = (a\ b)U^\top U(a\ b)^\top$, where matrix U depends on the point coordinates (x_i, y_i).

Solution.

$$E = \sum_{i=1}^{n}(ax_i+by_i-d)^2 = \sum_{i=1}^{n}(ax_i+by_i-a\overline{x}-b\overline{y})^2 = \sum_{i=1}^{n}(a(x_i-\overline{x})+b(y_i-\overline{y}))^2$$

$$E = \left|\left| \begin{bmatrix} x_1-\overline{x} & y_1-\overline{y} \\ \ldots & \ldots \\ x_n-\overline{x} & y_n-\overline{y} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \right|\right|^2 = (UN)^\top(UN)$$

where $N = \begin{bmatrix} a \\ b \end{bmatrix}$, $U = \begin{bmatrix} x_1-\overline{x} & y_1-\overline{y} \\ \ldots & \ldots \\ x_n-\overline{x} & y_n-\overline{y} \end{bmatrix}$.

4) Thus, the task is to minimize $||U(a\ b)^\top||$ under the constraint $a^2+b^2=1$. The solution for $(a\ b)^\top$ is the eigenvector of $U^\top U$ corresponding to the smallest eigenvalue, and d can be solved thereafter using the expression obtained above in the stage two.

Solution. This happens because the expression obtained in the previous section, is in quadratic form, so

$$N^\top U^\top UN$$

Then, when we have the corresponding eigenvector, d can be solved with the parameters a and b.

Exercise5

October 9, 2020

[1]:
```
# This cell is used for creating a button that hides/unhides code cells to␣
 ↪quickly look only the res.
# Works only with Jupyter Notebooks.

from IPython.display import HTML

HTML('''<script>
code_show=true;
function code_toggle() {
if (code_show){
$('div.input').hide();
} else {
$('div.input').show();
}
code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()"><input type="submit" value="Click here␣
 ↪to toggle on/off the raw code."></form>''')
```

[1]: <IPython.core.display.HTML object>

[2]:
```
# Description:
#   Exercise5 notebook.
#
# Copyright (C) 2018 Santiago Cortes,Juha Ylioinas
#
# This software is distributed under the GNU General Public
# Licence (version 2 or later); please refer to the file
# Licence.txt,included with the software,for details.

# Preparations
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

```python
# Select data directory
if os.path.isdir('/coursedata'):
    # JupyterHub
    course_data_dir = '/coursedata'
elif os.path.isdir('../../../coursedata'):
    # Local installation
    course_data_dir = '../../../coursedata'
else:
    # Docker
    course_data_dir = '/home/jovyan/work/coursedata/'

print('The data directory is %s' % course_data_dir)
data_dir = os.path.join(course_data_dir,'exercise-05-data/')
print('Data stored in %s' % data_dir)
```

```
The data directory is /coursedata
Data stored in /coursedata/exercise-05-data/
```

1 CS-E4850 Computer Vision Exercise Round 5

Remember to do the pen and paper assignments given in Exercise05task1.pdf.

The problems should be solved before the exercise session and solutions returned via MyCourses. Upload to MyCourses both: this Jupyter Notebook (.ipynb) file containing your solutions to the programming tasks and the exported pdf version of this Notebook file. If there are both programming and pen & paper tasks kindly combine the two pdf files (your scanned/LaTeX solutions and the exported Notebook) into a single pdf and submit that with the Notebook (.ipynb) file. Note that (1) you are not supposed to change anything in the utils.py and (2) you should be sure that everything that you need to implement should work with the pictures specified by the assignments of this exercise round.

1.1 Robust line fitting using RANSAC.

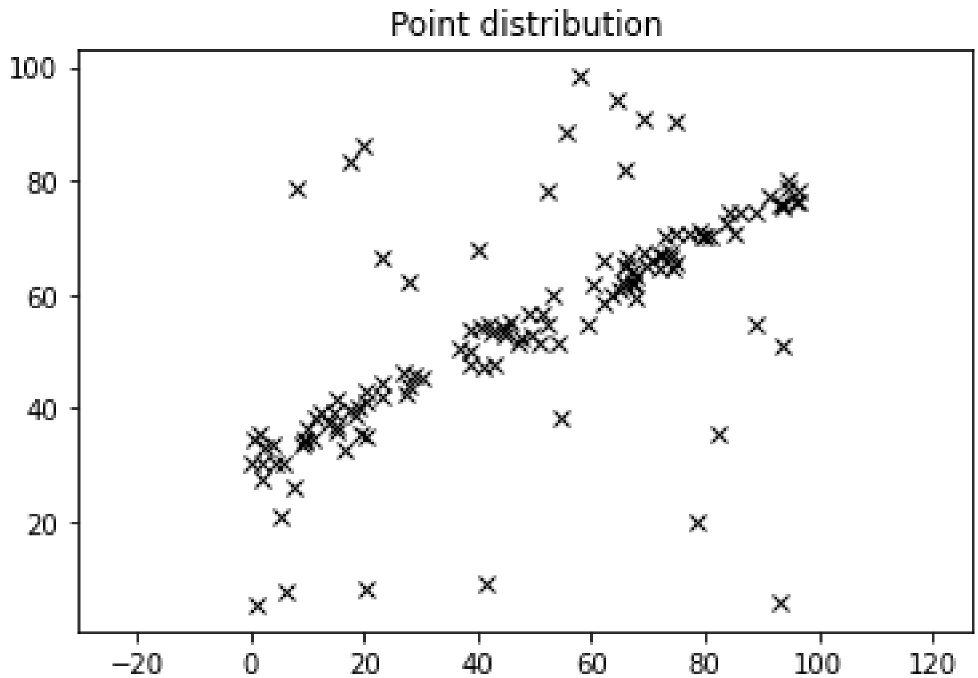
Run the example script robustLineFitting,which plots a set of points $(x_i, y_i), i = 1, \ldots, n$,and estimate a line that best fits to these points by implementing a RANSAC approach as explained in the slides of Lecture 4:

Repeat the following steps N times (set N large enough according to the guidelines given in the lecture):

- Draw 2 points uniformly at random from set (x_i, y_i).
- Fit a line to these 2 points.
- Determine the inliers to this line among the remaining points (i.e. points whose distance to the line is less than a suitably set threshold t).

Take the line with most inliers from previous stage and refit it using total least squares fitting to all inliers. Plot the estimated line and all the points (x_i, y_i) to the same figure and report the estimated values of the line's coefficients.

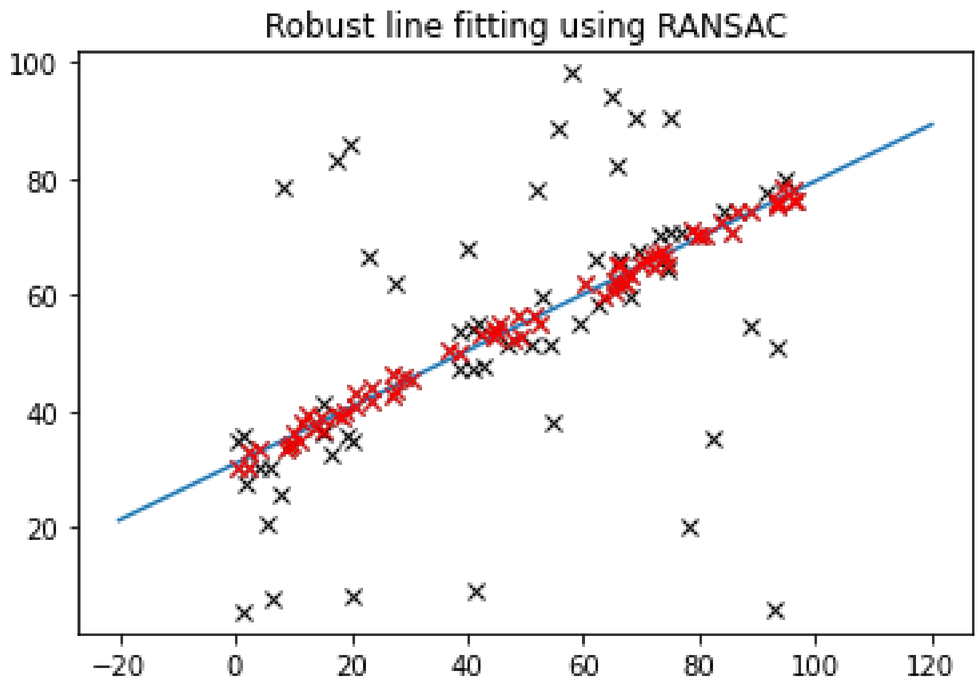
[3]:
```
# Load and plot points
data = np.load(data_dir+'points.npy')
x,y = data[0,:],data[1,:]
plt.plot(x,y,'kx')
plt.title('Point distribution')
plt.axis('equal')
plt.show()
```



[41]:
```
## Robust line fitting
##--your-code-starts-here--##
n = data.shape[1]
x_ = x.mean()
y_ = y.mean()

threshold = 2
p = 0.99
max_inliers = 0
res = []

i = 0
N = 1000000
while i < N:
    pts = np.random.choice(np.arange(n),size=2,replace=False)
```

```
    U = data[:,pts].T
    U[:,0] = U[:,0]-x_
    U[:,1] = U[:,1]-y_

    eig_val,eig_vec = np.linalg.eig(U.T@U)
    min_eig_val = np.argsort(eig_val)[0]

    a,b = eig_vec[:,min_eig_val]
    d = a*x_+b*y_

    dists = np.abs(a*x+b*y-d)

    inliers_i = dists < threshold
    inliers_x = x[inliers_i]
    inliers_y = y[inliers_i]
    n_inliers = np.sum(inliers_i)

    if n_inliers > max_inliers:
        max_inliers = n_inliers
        num = np.log(1-p)
        e = 1-n_inliers/n
        den = np.log(1-(1-e)**s)
        N = num/den

    res.append(np.array([a,b,n_inliers,inliers_x,inliers_y]))
    i += 1

res = np.array(res)

max_i = np.argmax(res[:,2])
_,_,n_inliers,inliers_x,inliers_y = res[max_i,:]
U_inliers = np.zeros((n_inliers,2))
U_inliers[:,0] = inliers_x-x_
U_inliers[:,1] = inliers_y-y_

eig_val,eig_vec = np.linalg.eig(U_inliers.T@U_inliers)
min_eig_val = np.argsort(eig_val)[0]

a,b = eig_vec[:,min_eig_val]
d = a*x_+b*y_

n_x = np.linspace(-20,120,10000)
n_y = (d-a*n_x)/b

plt.plot(x,y,'kx')
plt.plot(n_x,n_y)
plt.plot(inliers_x,inliers_y,"rx")
```

```
plt.title('Robust line fitting using RANSAC')
plt.axis('equal')
plt.show()
##--your-code-ends-here--##
```



1.2 Line detection by Hough transform. (Just a demo,no points given)

Run the example cell below,which illustrates line detection by Hough transform using opencv built-in functions.

[4]:
```
#DEMO CELL
# Logistic sigmoid function
def sigm(x):
    return 1 / (1+np.exp(-x))

# This demo detects the Canny edges for the input image,
# calculates the Hough transform for the Canny edge image,
# displays the Hough votes in an acculumator array
# and finally draws the detected lines

# Read image
I = cv2.imread(data_dir+'board.png',0)
r,c = I.shape

plt.figure(1)
```

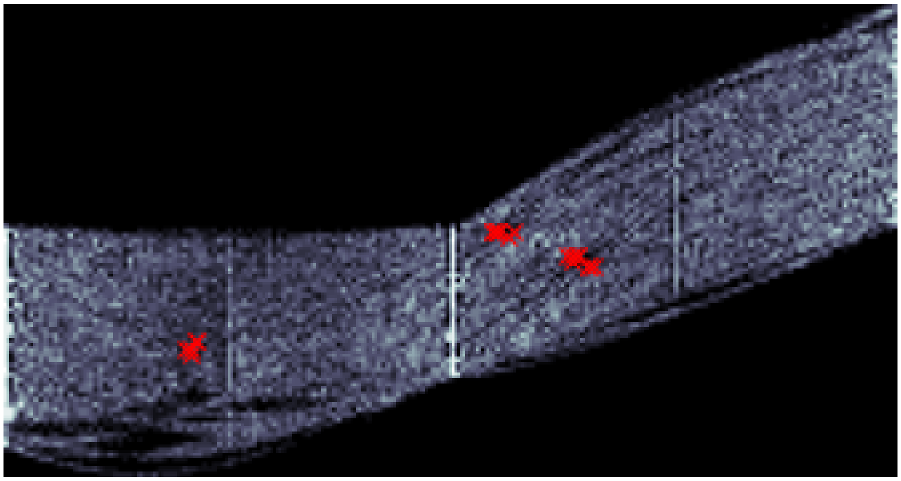
```python
plt.imshow(I,cmap='bone')
plt.title('Original image')
plt.axis('off')
# Find Canny edges. The input image for cv2.HoughLines should be
# a binary image,so a Canny edge image will do just fine.
# The Canny edge detector uses hysteresis thresholding,where
# there are two different threshold levels.
edges = cv2.Canny(I,80,130)
plt.figure(2)
plt.imshow(edges,cmap='gray')
plt.title('Canny edges')
plt.axis('off')
# Compute the Hough transform for the binary image returned by cv2.Canny
# cv2.HoughLines returns 2-element vectors containing (rho,theta)
# cv2.HoughLines(input image,radius resolution(pixels),angular resolution␣
↪(radians),treshold )
H = cv2.HoughLines(edges,0.5,np.pi/180,5)

# Display the transform
theta = H[:,0,1].ravel()
rho = H[:,0,0].ravel()

# Create an acculumator array and the bin coordinates for voting
x_coord = np.arange(0,np.pi,np.pi/180)
y_coord = np.arange(np.amin(rho),np.amax(rho)+1,(np.amax(rho)+1)/50)

acc = np.zeros([np.size(y_coord),np.size(x_coord)])

# Perform the voting
for i in range(np.size(theta)):
    x_id = np.argmin(np.abs(x_coord-theta[i]))
    y_id = np.argmin(np.abs(y_coord-rho[i]))
    acc[y_id,x_id] +=  1

# Pass the values through a logistic sigmoid function and normalize
# (only for the purpose of better visualization)
#acc = sigm(acc)
acc /= np.amax(acc)

plt.figure(3)
plt.imshow(acc,cmap='bone')
plt.axis('off')

plt.title('Hough transform space')

# Compute the Hough transform with higher threshold
# for displaying ~30 strongest peaks in the transform space
```

```python
H2 = cv2.HoughLines(edges,1,np.pi/180,150)

x2 = H2[:,:,1].ravel()
y2 = H2[:,:,0].ravel()

# Superimpose a plot on the image of the transform that identifies the peaks
plt.figure(3)
for i in range(np.size(x2)):
    x_id = np.argmin(abs(x_coord-x2[i]))
    y_id = np.argmin(abs(y_coord-y2[i]))
    plt.plot(x_id,y_id,'xr','Linewidth',0.1)

# Visualize detected lines on top of the Canny edges.
plt.figure(4)
plt.imshow(I,cmap='bone')
plt.title('Detected lines')
plt.axis('off')


for ind in range(0,len(H2)):
    line=H2[ind,0,:]
    rho=line[0]
    theta=line[1]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0+1000*(-b))
    y1 = int(y0+1000*(a))
    x2 = int(x0-1000*(-b))
    y2 = int(y0-1000*(a))

    plt.plot((x1,x2),(y1,y2))

#plt.plot(xk,yk,'m-')
plt.xlim([0,np.size(I,1)])
plt.ylim([0,np.size(I,0)])
plt.show()
```

Original image

Canny edges

Hough transform space



Detected lines