Aalto-yliopisto
Aalto-universitetet
Aalto University

CS-E4850
Computer Vision

Homework 5

Least Squares

Aitor Urruticoechea Puig
aitor.urruticoecheapuig@aalto.fi
Student N°101444219
October 2024

Notice

This is only one half of the total whole of this round of homework exercise, as programming tasks have been completed in a separate Jupyter Notebook. Please do refer to that document for the rest of the exercise(s).

Contents

Exercise 1: Total least squares line fitting

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:

1.1 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 $|ax_i + by_i - d|$.

Let us begin with the classical formula of the distance point-to-line:

$$D = \frac{|Ax_i + By_i + C|}{\sqrt{A^2 + B^2}} \tag{1}$$

where A, B, and C are the coefficients of the classical line equation $Ax + By + C = 0$ and (x_i, y_i) is the point. This can be now normalized to follow the constraints given, namely $ax + by - d = 0$ for the line equation and the normalization $a^2 + b^2 = 1$. This means that the following transformation will be needed:

$$\begin{cases} A = a \\ B = b \\ C = -d \end{cases} \tag{2}$$

which, with the normalization $a^2 + b^2 = 1$, results in:

$$D = \frac{|ax_i + by_i - d|}{\sqrt{a^2 + b^2}} = |ax_i + by_i - d| \tag{3}$$

Thus reaching the objective equation.

1.2 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* $\delta E/\delta d$*, set it to zero, and solve* d *in terms of* a *and* b*.*

To start with, one can diffferentiate E with repspect to d as hinted:

$$\frac{\delta E}{\delta d} = \frac{\delta}{\delta d}\sum_{i=1}^{n}(ax_i + by_i - d)^2 = -2\sum_{i=1}^{n}(ax_i + by_i - d) \tag{4}$$

Finding the minimum can be done by zeroing-out the partial derivative:F

$$0 = -2\sum_{i=1}^{n}(ax_i + by_i - d) = \sum_{i=1}^{n}(ax_i + by_i - d) \tag{5}$$

Now, solving for d:

$$0 = \sum_{i=1}^{n}(ax_i + by_i - d) = \sum_{i=1}^{n}(ax_i + by_i) - \sum_{i=1}^{n} d \tag{6}$$

$$d = \frac{1}{n}\sum_{i=1}^{n}(ax_i + by_i) \tag{7}$$

Finally getting the optimized equation.

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

Let us start by substituting the obtained d expression in E:

$$E = \sum_{n}^{i=1}\left(ax_i + by_i - \frac{1}{n}\sum_{n}^{j=1}(ax_j + by_j)\right)^2 = \sum_{n}^{i=1}\left[ax_i + by_i - \frac{a}{n}\sum_{n}^{j=1}x_j - \frac{b}{n}\sum_{n}^{j=1}y_j)\right]^2 \tag{8}$$

by defining $\bar{x} = \frac{1}{n}\sum_{n}^{j=1}x_j$ and $\bar{y} = \frac{1}{n}\sum_{n}^{j=1}y_j$:

$$E = \sum_{n}^{i=1}[a(x_i - \bar{x}) + b(y_i - \bar{y})]^2 = \sum_{n}^{i=1}a^2(x_i - \bar{x})^2 + b^2(y_i - \bar{y})^+ 2ab(x_i - \bar{x})(y_i - \bar{y}) \tag{9}$$

Now, with this expanded form, one can define the matrix U:

$$U = \begin{bmatrix} x_1 - \bar{x_1} & y_1 - \bar{y_1} \\ x_2 - \bar{x_2} & y_2 - \bar{y_2} \\ \ldots & \ldots \end{bmatrix} \tag{10}$$

If one is to now rewrite E, the desired expression appears:

$$E = (a\ b)U^TU(a\ b)^T \tag{11}$$

where (a b is the row vector of coefficients, and U^TU is the 2×2 matrix obtained by multiplying the transpose of U with U.

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

Minimizing $\left[U(a\ b)^T\right]^2$ with the constraint $a^2 + b^2 = 1$ is equivalent to finding the vector $(a\ b)^T$ that minimizes the Rayleigh quotient:

$$\frac{(a\ b)U^TU(a\ b)^T}{a^2 + b^2} = (a\ b)U^TU(a\ b)^T \tag{12}$$

The minimization of this quadratic form leads to solving the eigenvalue problem for the matrix U^TU. The eigenvectors obtained, thus, represent the possible solutions for $(a\ b)U^TU(a\ b)^T$, and the corresponding eigenvalues give the values of $\left[U(a\ b)^T\right]^2$. To minimize this, in turn, one needs to choose the eigenvector of U^TU corresponding to the smallest eigenvalue. Once this is done, it is just a matter to solve for d in equation 7. This gives the optimal value of d in terms of the point coordinates and the coefficients a and b that minimize the sum of squared distances.

Exercise5

October 3, 2024

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

```
# 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:

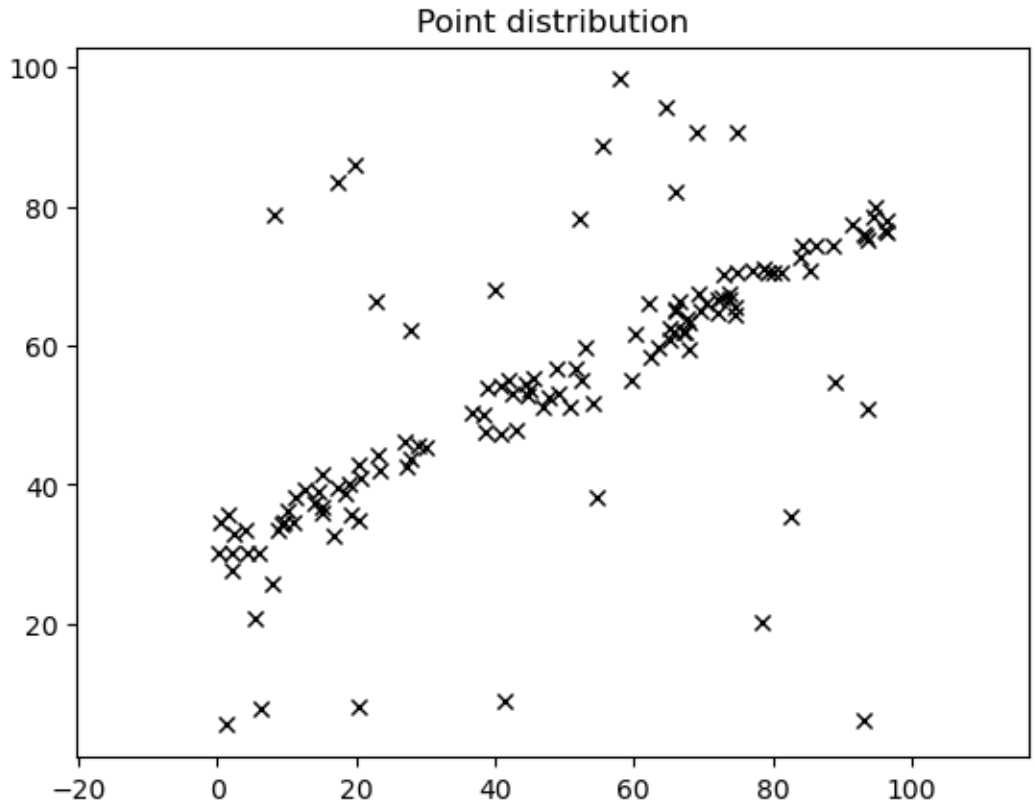
Repeat the following steps N times (set N large enough according to the guidelines given in the lecture):

- 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

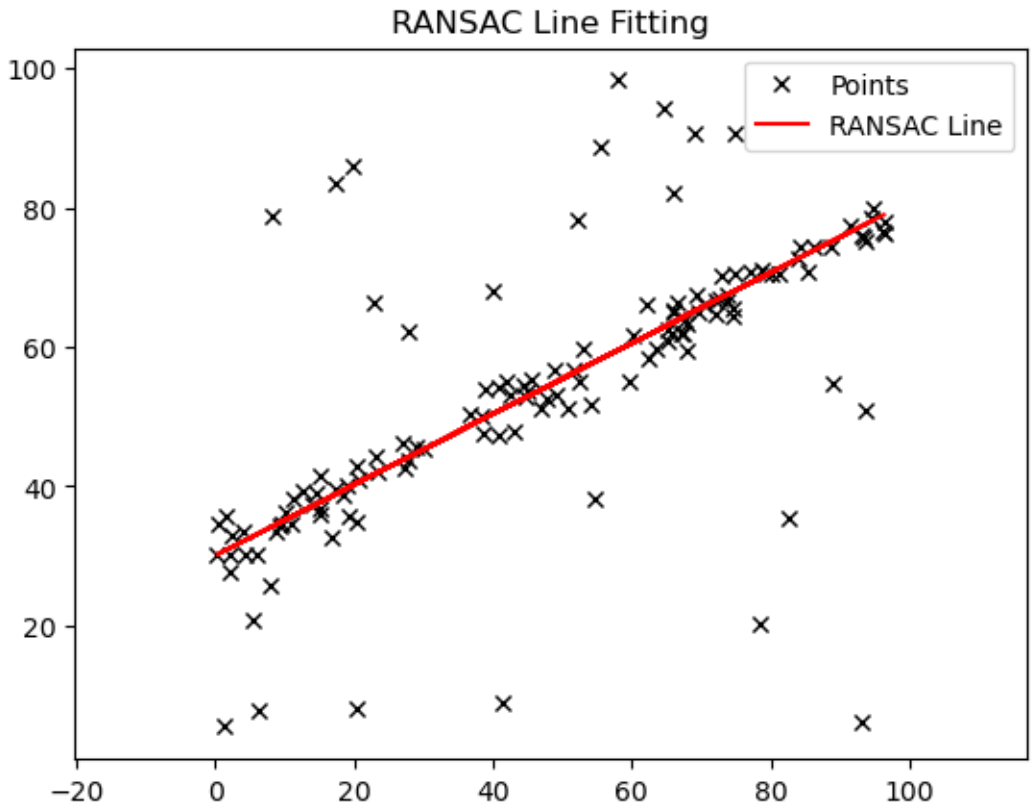
estimated values of the line's coefficients.

[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()
```



[4]:
```
## Robust line fitting
##--your-code-starts-here--##
def point_to_line_dist(x, y, m, c):
    return abs(y - (m * x + c)) / np.sqrt(m**2 + 1)

# Parameters
N = 1000  # Iterations
t = 1.0   # Distance threshold
best_m, best_c = None, None
max_inliers = 0
best_inliers = []
```

```python
# RANSAC
for _ in range(N):
    # Random two points
    idx = np.random.choice(len(x), 2, replace=False)
    x1, y1 = x[idx[0]], y[idx[0]]
    x2, y2 = x[idx[1]], y[idx[1]]

    # Fit line
    if x2 == x1:  # No division by zero lol
        continue
    m = (y2 - y1) / (x2 - x1)
    c = y1 - m * x1

    # Inliners
    inliers = []
    for i in range(len(x)):
        if point_to_line_dist(x[i], y[i], m, c) < t:
            inliers.append(i)

    # Most inliers are kept
    if len(inliers) > max_inliers:
        max_inliers = len(inliers)
        best_m, best_c = m, c
        best_inliers = inliers

# Refit line
x_inliers = x[best_inliers]
y_inliers = y[best_inliers]
A = np.vstack([x_inliers, np.ones(len(x_inliers))]).T
best_m, best_c = np.linalg.lstsq(A, y_inliers, rcond=None)[0]

# Plots
plt.plot(x, y, 'kx', label='Points')
plt.plot(x_inliers, best_m * x_inliers + best_c, 'r-', label='RANSAC Line')
plt.legend()
plt.title('RANSAC Line Fitting')
plt.axis('equal')
plt.show()

print(f"Estimated line: y = {best_m:.2f}x + {best_c:.2f}")

##--your-code-ends-here--##
```




```
Estimated line: y = 0.51x + 30.01
```

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.

[5]:
```
#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)
```

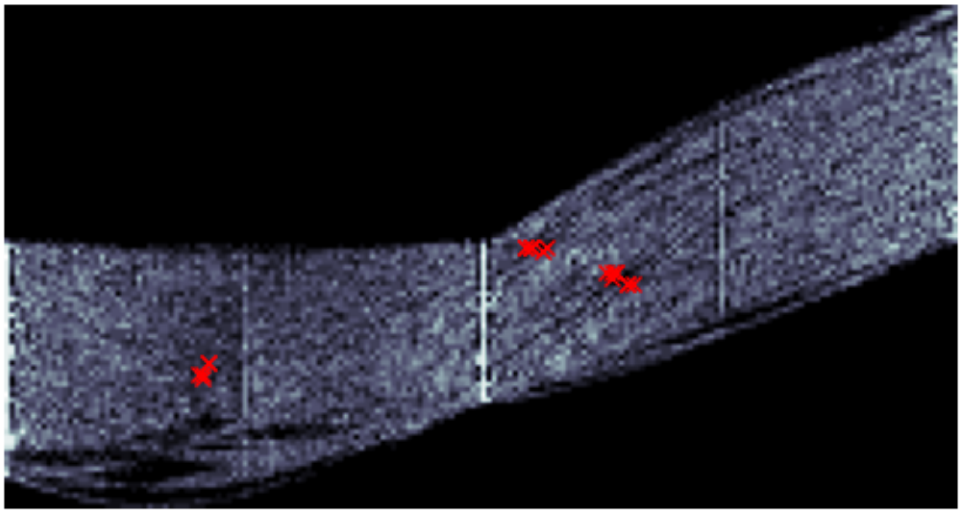
```
plt.imshow(I, cmap='bone')
plt.title('Original image')
plt.axis('off')
# Find Canny edges. The input image for cv2.HoughLines should be
# a binary image, so a Canny edge image will do just fine.
# The Canny edge detector uses hysteresis thresholding, where
# there are two different threshold levels.
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
```

```
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

[ ]: