DATA.ML.300 Daniel Kusnetsoff

Ex. round 6.

Task 1. Fundamental matrix and essential matrix

a/ Fundamental matrix is 3x3 matrix rank2. It is used for defining points in space from a different view. If x=first view x'=second view, image points give relation $x'^T F x = 0$.
Essential matrix is a specialized case of fundamental matrix, using normalized image coordinates. Compared to the fundamental matrix, the essential matrix has less degrees of freedom and some additional properties such as the normalized coordinates.

b/ $E = [t]_\times R = R[R^T t]_\times$ (fundamental matrix corresponding to the pair of normalized cameras)

Essential matrix

$\hat{x}'^T E \hat{x} = 0$ (substitution of $\hat{x}$ and $\hat{x}' \Rightarrow x'^T K'^{-T} E K^{-1} x = 0$)

$E = K'^T F K$ (needed normalized camera matrix) $K^{-1} P = [R|t]$

c) A 3x3 homogenous matrix has nine elements.
→ It has 8 independent ratios.
As the fundamental matrix is according to the constraint F=0, one degree is removed
→ 7 degrees of freedom.

d) The essential matrix has five degrees of freedom.
- Rotation matrix : 3
- Translation : 3

As the essential matrix is homogenous and there is scale ambiguity → 5 degrees of freedom.

# ExRound6

February 20, 2022

```
[ ]: %task2 Matlab
```

```
[ ]: ##cam_calibration.m

%% Calibration of the first camera
% The given image coordinates were originally localized manually using
im1=imread('im1.jpg');

x1 = 1.0e+03 * [0.7435; 3.3315; 0.8275; 3.2835;
                0.5475; 3.9875; 0.6715; 3.8835];

y1 = 1.0e+03 * [0.4455; 0.4335; 1.7215; 1.5615;
                0.3895; 0.3895; 2.1415; 1.8735];

% Image coordinates of points as rows of matrix 'abcdefgh'
abcdefgh=[x1(:) y1(:)];

% World coordinates of the points (dimensions of the shelf)
ABCDEFGH=[758  0 -295;
          0  0 -295;
          758  360 -295;
          0  360 -295;
          758  0 0;
          0  0 0;
          758  360 0;
          0  360 0];

% Plot manualy localized points
figure;imshow(im1);hold on
title('Cyan: manually located points  Red: projected points')
plot(x1,y1,'c+','MarkerSize',10);
labels={'a','b','c','d','e','f','g','h'};
for i=1:length(x1)
    ti=text(x1(i),y1(i),labels{i});
    ti.Color='cyan';
    ti.FontSize=20;
end
```

```
%% Your task is to implement the missing function 'camcalibDLT.m'.
% The algorithm is summarised on the lecture slides and exercise sheet.
% The function takes the homogeneous coordinates of the points as input.
P1 = camcalibDLT([ABCDEFGH ones(8,1)], [abcdefgh ones(8,1)]);

% Check the results by projecting the world points with the estimated P
% the projected points should overlap with manually localized points
pproj1 = P1*[ABCDEFGH';ones(1,8)];
for i=1:8
    plot(pproj1(1,i)/pproj1(3,i),pproj1(2,i)/pproj1(3,i),'rx','MarkerSize',20);
end


%% Calibration of the second camera
im2 = imread('im2.jpg');

x2 = 1.0e+03 * [0.5835; 3.2515; 0.6515; 3.1995;
               0.1275; 3.7475; 0.2475; 3.6635];

y2 = 1.0e+03 * [0.4135; 0.4015; 1.6655; 1.5975;
                0.3215; 0.3135; 2.0295; 1.9335];

figure;imshow(im2);hold on
title('Cyan: manually located points  Red: projected points')
plot(x2,y2,'c+','MarkerSize',10);
labels={'a','b','c','d','e','f','g','h'};
for i=1:length(x2)
    ti=text(x2(i),y2(i),labels{i});
    ti.Color='cyan';
    ti.FontSize=20;
end
%% Second camera projection matrix
P2 = camcalibDLT([ABCDEFGH ones(8,1)], [x2 y2 ones(8,1)]);

% Again, check results by projecting world coordinates with P2
pproj2 = P2*[ABCDEFGH';ones(1,8)];
for i=1:8
    plot(pproj2(1,i)/pproj2(3,i),pproj2(2,i)/pproj2(3,i),'rx','MarkerSize',20);
end
```

[ ]:

```
##camcalibDLT.m

function P=camcalibDLT(x_world, x_im)
    % Inputs:
    %   x_world: World coordinates with shape (point_id, coordinates)
    %   x_im: Image coordinates with shape (point_id, coordinates)
    %
    % Outputs:
    %   P: Camera projection matrix with shape (3,4)

    % Create the matrix A
    %%-your-code-starts-here-%%
    n = 8;
    A = zeros(16,12);
    y = x_im(:,2);
    x = x_im(:,1);
    for i=1:n
        A(i*2-1,:) = [zeros(1,4) x_world(i,:) -y(i)*x_world(i,:)];
        A(i*2,:) = [x_world(i,:) zeros(1,4) -x(i)*x_world(i,:)];
    end
    disp(A)
    %%-your-code-ends-here-%%

    % Perform homogeneous least squares fitting.
    % The best solution is given by the eigenvector of
    % A.T*A with the smallest eigenvalue.
    %%-your-code-starts-here-%%
    eigvs=A'*A;
    [V,D] = eig(eigvs)
    [~,ind]= min(diag(D));
    ev= V(:,ind);
    %%-your-code-ends-here-%%

    % Reshape the eigenvector into a projection matrix P
    P = (reshape(ev,4,3))'; % here ev is the eigenvector above
    %P = [0 0 0 0;0 0 0 0;0 0 0 1];  % remove this and uncomment the above
end
```

[ ]:
```
Task 3 - Python
```

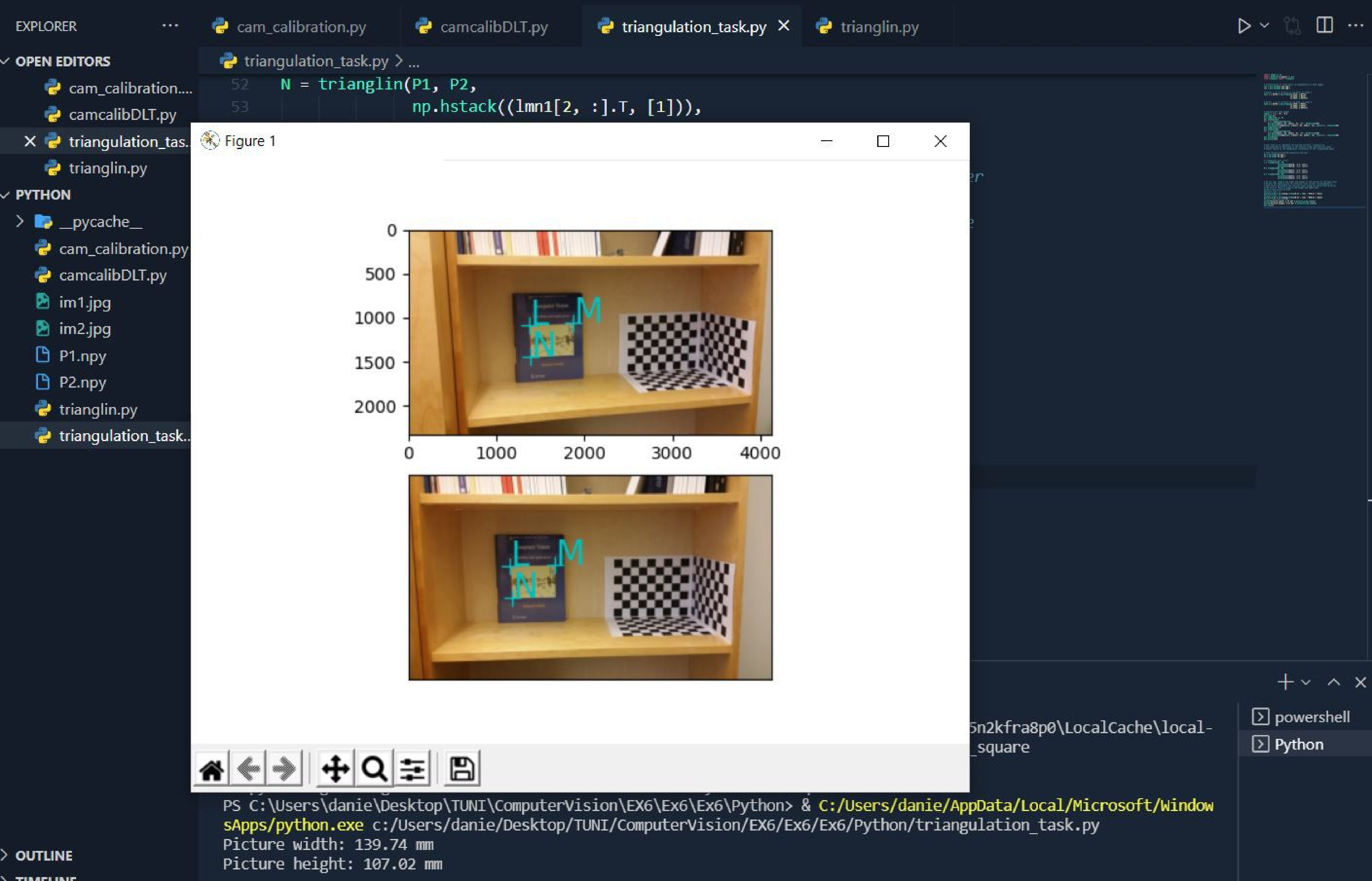
[ ]:
```
##triangulation_task.py

import numpy as np
import matplotlib.pyplot as plt
from trianglin import trianglin


# Visualization of three point correspondences in both images
im1 = plt.imread('im1.jpg')
im2 = plt.imread('im2.jpg')

# Points L, M, N (corners of the book) in image 1
```

```python
lmn1 = 1.0e+03 * np.array([[1.3715, 1.0775],
                           [1.8675, 1.0575],
                           [1.3835, 1.4415]])

# Points L, M, N (corners of the book) in image 2
lmn2 = 1.0e+03 * np.array([[1.1555, 1.0335],
                           [1.6595, 1.0255],
                           [1.1755, 1.3975]])

# Annotate and show images
labels = ['L', 'M', 'N']
plt.figure()
plt.subplot(2, 1, 1)
plt.imshow(im1)
for i in range(len(labels)):
    plt.plot(lmn1[i, 0], lmn1[i, 1], 'c+', markersize=10)
    plt.annotate(labels[i], (lmn1[i, 0], lmn1[i, 1]), color='c', fontsize=20)
plt.subplot(2,1,2)
plt.imshow(im2)
for i in range(len(labels)):
    plt.plot(lmn2[i, 0], lmn2[i, 1], 'c+', markersize=10)
    plt.annotate(labels[i], (lmn2[i, 0], lmn2[i, 1]), color='c', fontsize=20)
plt.xticks([])
plt.yticks([])


# Your task is to implement the missing function 'trianglin.py'
# The algorithm is described in the lecture slides and exercise sheet.
# Output should be the homogeneous coordinates of the triangulated point.

# Load the pre-calculated projection matrices
P1 = np.load('P1.npy')
P2 = np.load('P2.npy')

# Triangulate each corner
L = trianglin(P1, P2,
              np.hstack((lmn1[0, :].T, [1])),
              np.hstack((lmn2[0, :].T, [1])))
M = trianglin(P1, P2,
              np.hstack((lmn1[1, :].T, [1])),
              np.hstack((lmn2[1, :].T, [1])))
N = trianglin(P1, P2,
              np.hstack((lmn1[2, :].T, [1])),
              np.hstack((lmn2[2, :].T, [1])))

# We can then compute the width and height of the picture on the book cover
# Convert the above points to cartesian, form vectors corresponding to
# book covers horizontal and vertical sides using the points and calculate
# the norm of these to acquire the height and width (mm).
##-your-code-starts-here-##
#picture_w_mm = 0.
picture_w_mm = np.linalg.norm(L[0:3] / L[3] - M[0:3] / M[3])
#picture_h_mm = 0.
picture_h_mm = np.linalg.norm(L[0:3] / L[3] - N[0:3] / N[3])
##-your-code-ends-here-##
print("Picture width: %.2f mm" % picture_w_mm.item())
print("Picture height: %.2f mm" % picture_h_mm.item())
plt.show()


## Picture width : 139.74
## Picture height : 107.20
```

[ ]:
```
##trianglin.py


import numpy as np


def trianglin(P1, P2, x1, x2):
    """
    :param P1: Projection matrix for image 1 with shape (3,4)
    :param P2: Projection matrix for image 2 with shape (3,4)
    :param x1: Image coordinates for a point in image 1
    :param x2: Image coordinates for a point in image 2
    :return X: Triangulated world coordinates
    """

    # Form A and get the least squares solution from the eigenvector
    # corresponding to the smallest eigenvalue
    ##-your-code-starts-here-##
    x1_a = np.array([[0,-1*x1[2],x1[1]],[x1[2],0,-1*x1[0]],[-1*x1[1],x1[0],0]])
    x2_a = np.array([[0,-1*x2[2],x2[1]],[x2[2],0,-1*x2[0]],[-1*x2[1],x2[0],0]])

    x1_c = np.dot(x1_a,P1)
    x2_c = np.dot(x2_a,P2)
    A = np.concatenate((x1_c,x2_c))
    #did not get implementation with np.linalg.eig to work -> this works
    u,s,w=np.linalg.svd(A)
    smallest=np.argmin(s)
    X = w[smallest]


    ##-your-code-ends-here-##
    #X = np.array([0, 0, 0, 1])  # remove me

    return X
```

[ ]:

```
```



```
N = trianglin(P1, P2,
              np.hstack((lmn1[2, :].T, [1])),
```

```
5n2kfra8p0\LocalCache\local-
_square

PS C:\Users\danie\Desktop\TUNI\ComputerVision\EX6\Ex6\Ex6\Python> & C:/Users/danie/AppData/Local/Microsoft/Window
sApps/python.exe c:/Users/danie/Desktop/TUNI/ComputerVision/EX6/Ex6/Ex6/Python/triangulation_task.py
Picture width: 139.74 mm
Picture height: 107.02 mm
```