תרגיל 3

מגישים:

ארז איתמר

ת.ז:308430115

זילברשטיין אריאל

ת.ז:315314799

Targil 2.5

Part 1 : Compute the fundamental matrix F ¶

Compute the fundamental matrix F from a pair of normalized*

images with at least 8 corresponding points

***** You can do your calculations “manually” or you can code it

***** however you choose to solve; show it in your submitted

solution: calculations/code snippet/pseudo-code

Imports

In [4]:

```
import numpy as np
from scipy.misc import imread
import matplotlib.pyplot as plt
import imageio
```

Code for regular fundemental matrix calulcations

In [5]:

```
def find_fundamental_matrix(cords_1, cords_2):
    ''''
    Section (ii) of the The normalized 8-point algorithm seen at
    page 279 in Multiple View Geometry in Computer Vision,second edition

    Finds the fundamental matrix F̂'
    :return The fundamental matrix F̂'
    '''

    # the total amount of corresponding points
    total_points = cords_2.shape[0]


    # see equation (11.3) at page 279 in Multiple View Geometry in Computer Visi
on,second edition
    A = np.zeros((total_points, 9))

    for i in range(total_points):
        A[i] = np.array([
            cords_1[i][0]*cords_2[i][0],cords_2[i][0]*cords_1[i][1],cords_2[i][0
],
            cords_2[i][1]*cords_1[i][0],cords_1[i][1]*cords_2[i][1],
            cords_2[i][1], cords_1[i][0],cords_1[i][1], 1
        ])

    # linear solution
    U, D, V = np.linalg.svd(A, full_matrices=True)

    f = V[-1, :]

    F_hat = np.reshape(f, (3, 3))
    U, s_hat, V = np.linalg.svd(F_hat, full_matrices=True)
    s_hat[::-1].sort()
    S =np.diag(s_hat)
    S[2][2] = 0

    # constraint enforcement per section (b) in the algorithm in page 279 of
    # Multiple View Geometry in Computer Vision,second edition
    F = np.dot(U, np.dot(S, V))

    return F
```

Code for normalized fundamental matrix calculations:

In [6]:

```
def normalization(x):
    '''
    Section (i) of the The normalized 8-point algorithm seen at
    page 279 in Multiple View Geometry in Computer Vision,second edition

    Transform the image coordinates according to x̂=Tx and x̂'=T'x' where T and T'
are normalizing transformation
     consisting of translation and scaling.
    :param cords_1: x coordinate set
    :param cords_2: x'coordinate set
    :return:
    T,T', x̂,x̂'
    '''

    cords=x[:, 0:2]
    centroid = np.mean(cords,axis=0)
    centered=cords-centroid
    dists=np.sqrt(np.sum((centered)**2,axis=1))
    mean_dist=np.mean(dists,axis=0)

    norm_mat=np.array([
         [np.sqrt(2)/mean_dist, 0, -1*np.sqrt(2)/(mean_dist)*centroid[0]],
                    [0, np.sqrt(2)/mean_dist,-1*np.sqrt(2)/(mean_dist)*centroid[
1]],
                    [0, 0, 1]
    ])
    transformed_cords = norm_mat.dot(x.T).T

    return  transformed_cords,norm_mat
```

In [7]:

```
def de_normalization(trans_mat_1, trans_mat_2,F_hat):
    '''
        Section (iii) of the The normalized 8-point algorithm seen at
        page 279 in Multiple View Geometry in Computer Vision,second edition

        Set F = (T')^T * F̂ * T.
        Matrix F is the fundamental matrix corresponding
        to the original data
    :param trans_mat_1:
    :param trans_mat_2:
    :param Fq:
    :return:
    '''
    return trans_mat_2.T.dot(F_hat).dot(trans_mat_1)
```

In [8]:

```
def normalized_eight_point_algorithm(points_1, points_2):

    trans_cords_1,trans_mat_1=normalization(points_1)
    trans_cords_2,trans_mat_2=normalization(points_2)


    # find the matrix F̂ corresponding to the matches x̂ and x̂'
    F = find_fundamental_matrix(trans_cords_1, trans_cords_2)

    # set F=(T')^T * F̂' * T
    normalized_F = de_normalization(trans_mat_1,trans_mat_2,F)

    return normalized_F
```

Part 2 : Draw the epipolar lines on both images

Helper methods:

In [9]:

```
def homogenous_coordinates(cord):
        return np.array([cord[0], cord[1], 1])
```

In [10]:

```
def get_line(F,cord,sx,sy):
        v = homogenous_coordinates(cord)
        l = F.dot(v)
        s = np.sqrt(l[0]**2+l[1]**2)
        l = l/s

        # get start and end points according to image limit
        if l[0] != 0:
            ye = sy-1
            ys = 0
            xe = -(l[1] * ye + l[2])/l[0]
            xs = -(l[1] * ys + l[2])/l[0]
        else:
            xe = sx-1
            xs = 0
            ye = -(l[0] * xe + l[2])/l[1]
            ys = -(l[0] * xs + l[2])/l[1]

        return xs,xe,ys,ye
```

In [15]:

```
def plot_epipolar_lines_on_images(cords, im1, im2, F):
    sy, sx, _ = im2.shape
    f, [ax1, ax2] = plt.subplots(1, 2, figsize=(12, 9))
    ax1.imshow(im1)
    ax1.set_title('Image With points')
    ax2.imshow(im2)
    ax2.set_title('Image with corresponding epipolar lines')
    axes = plt.gca()
    axes.set_xlim([0,im2.shape[1]])
    axes.set_ylim([im2.shape[0],0])

    for cord in cords:
        # ge line cordinates
        xs,xe,ys,ye=get_line(F,cord,sx,sy)

        # plot line
        ax2.plot([xs, xe], [ys, ye], linewidth=2)

        # plot corresponding points
        ax1.plot(cord[0], cord[1], '*', MarkerSize=6, linewidth=2)

    plt.show()
```

In [12]:

```
def plot_epipolar_lines_on_images(cords, im1, im2, F):
    sy, sx, _ = im2.shape
    f, [ax1, ax2] = plt.subplots(1, 2, figsize=(12, 9))
    ax1.imshow(im1)
    ax1.set_title('Image A')
    ax2.imshow(im2)
    ax2.set_title('Image B')
    axes = plt.gca()
    axes.set_xlim([0,im2.shape[1]])
    axes.set_ylim([im2.shape[0],0])

    for cord in cords:
        # ge line cordinates
        xs,xe,ys,ye=get_line(F,cord,sx,sy)

        # plot line
        ax2.plot([xs, xe], [ys, ye], linewidth=2)

        # plot corresponding points
        ax1.plot(cord[0], cord[1], '*', MarkerSize=6, linewidth=2)

    plt.show()
```

Sample

In [16]:

```
def example1():
        # Read in the data
    im1 = imageio.imread('1.jpeg')
    im2 = imageio.imread('2.jpeg')

    cords_1 = np.array([
        [592, 351, 1],
        [616, 346, 1],
        [633, 339, 1],
        [648, 335, 1],
        [592, 265, 1],
        [638, 263, 1],
        [387, 298, 1],
        [504, 266, 1],

    ])

    cords_2 = np.array([
        [623, 474, 1],
        [760, 438, 1],
        [846, 414, 1],
        [902, 401, 1],
        [834, 251, 1],
        [928, 243, 1],
        [444, 314, 1],
        [661, 261, 1],
    ])
    F=normalized_eight_point_algorithm(cords_1,cords_2)
    plot_epipolar_lines_on_images(cords_1,im1,im2,F)
```

display

In [17]:

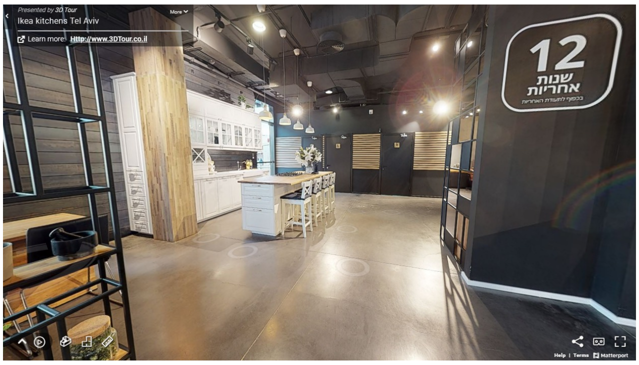
```
#
example1()
```

In [ ]:

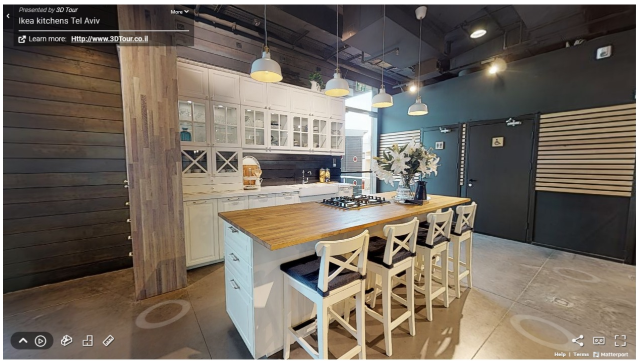
```

```

תמונות הקלט:

תמונה : 1



תמונה : 2



$PART\ 2$

$show\ how\ the\ linear\ method\ extends\ to\ n > 2\ images$

נוכיח:

נניח כי קיימות n תמונות שנלקחו בn זוויות שונות.

לכול תמונה i נסמן את התצפיות :

$$u_i = P_i\widehat{X}$$

כאשר $\widehat{X} = \left(X; 1\right)$ נתון ויהיו $p_i^{1T}, p_i^{2T}, p_i^{3T}$ השורות של p_i ו x_i, y_i הקואורדינטות של u_i.

אז נקבל כי מתקיים:

$$\left(u_i\right) \ \times \ \left(P_i\widehat{X}\right) = 0$$

מכאן:

$$x_i\left(p_i^{3T}\widehat{X}\right) - \left(p_i^{1T}\widehat{X}\right) = 0$$

$$y_i\left(p_i^{3T}\widehat{X}\right) - \left(p_i^{2T}\widehat{X}\right) = 0$$

$$x_i\left(p_i^{2T}\widehat{X}\right) - y\left(p_i^{1T}\widehat{X}\right) = 0$$

נשים לב כי בדומה למקרה של $n = 2$ שמופיע בעמוד 312 בספר $Multiple\ View\ Geometry$

המשוואות שקיבלנו הינן ליניאריות בקומפוננטות של $\widehat{X}$.

מכאן מספיק לקחת שתי משוואת למשל:

$$x_i\left(p_i^{3T}\widehat{X}\right) - \left(p_i^{1T}\widehat{X}\right) = 0$$

$$y_i\left(p_i^{3T}\widehat{X}\right) - \left(p_i^{2T}\widehat{X}\right) = 0$$

מאחר ויש nתמונות בn זוויות שונות נקבל כי סך הכול כי בידינו $2n$ משוואות ליניאריות לצורך החישוב של X בבעיית ה$triangulation$.

נגדיר:

$$A = \begin{pmatrix} x_1\left(p_1^{3T}\right) - \left(p_1^{1T}\right) = 0 \\ y_1\left(p_1^{3T}\right) - \left(p_1^{2T}\right) = 0 \\ . \\ . \\ . \\ x_n\left(p_n^{3T}\right) - \left(p_n^{1T}\right) = 0 \\ y_n\left(p_n^{3T}\right) - \left(p_n^{2T}\right) = 0 \end{pmatrix}$$

ונקבל את התנאי הבא:

$AX = 0$

פיתרון של מערכת משוואות מהצורה $AX = 0$ הוצג בעמוד 90בספר
$Multiple\ View\ Geometry$

אז הפתרון נעשה על ידי שנקבע את הקורדינטה האחרונה של $\widehat{X}$ בתור 1 ואז הפתרון $AX = 0$

יצטמצם למערכת משוואות של $2n$

נפעיל את שיטת ה $linear\ least\ square\ solution$

או שנפעיל את שיטת DLT על ידי $Singular\ value\ Decomposition$ שכמתואר

בספר $Multiple\ View\ Geometry$ בהכרח ייתן לי את הפתרון האופטימלי.

סך הכול לא משנה באיזה שיטה נבחר ,נקבל את הפתרון האופטימלי כנדרש!

12.8.2(i)

A method for triangulation in the case of pure translational

We will suggest a method for triangulation in the case of pure translational motion of the cameras.

For this motion the fundamental matrix is skew symmetric with only two degrees of freedom [page 249]. We will try to triangulate a given point X in the 3-space by having a set of n corresponding points and their projection onto 2-space's $\{(x_i, y_i)\}^n$ we will get that that for all pairs of points from the given set the "real" epipoler lines e and e' are the same(e = e') .

We will consider a parametrization of the angle between the image X axes and the epipoler line, as θ, and we will consider the origin of exes to be in the epipole.

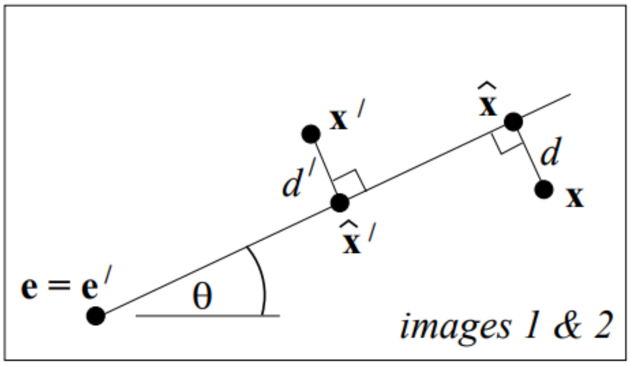


Fig. 12.9

The method we will suggest is choosing the line e (the epipoler line) as the line containing e which minimizes the perpendicular distance from each point from the set of the n matching points $\{(x_i, y_i)\}^n$to the epipolar suggested line. Let $r_i = \left\|\begin{matrix} x_i \\ y_i \end{matrix}\right\|$ and $\theta_i = \arctan\left(\frac{y_i}{x_i}\right)$

Then the close formula will be:

$$find\ argmin_\theta \sum_{(x_i, y_i)} \sqrt{{r_i}^2 - {r_i}^2 \cos(\theta - \theta_i)}$$

Then we can project the given points on the epipoler line. So it guarantees that the two lines from the centers of the cameras to the corresponding points will intercept in certain point in the 3-space (the two lines are on the same plane which is determined by the epipoler line) this interception point will be the X we were looking for.

In particular, this method is valid for the two images case as seen in Fig. 12.9